OIPE JC19
NOV 17 2003
PATENT & TRADEMARK OFFICE

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Appl. No.	:	10/659,161	Confirmation No. To be assigned.
Applicant	:	Thomas Bennett et al.	
Filed	:	September 10, 2003	
TC/A.U.	:	To be assigned	
Examiner	:	To be assigned	
Docket No.	:	RBI0004	
Customer No.	:	27268	

CLAIM FOR PRIORITY

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

Applicant hereby claims foreign priority benefits under Title 35, United States Code Section 119, of Canadian Application No. 2,402,761 filed on September 10, 2002. A certified copy of the priority document is enclosed herewith.

In the event the Applicant has overlooked the need for an extension of time, an additional extension of time, payment of fee, or additional payment of fee, Applicant hereby conditionally petitions therefor and authorizes that any charges be made to Deposit Account No. 02-0390, Baker & Daniels.

Should any questions concerning any of the foregoing arise, the Examiner is invited to telephone the undersigned at (317) 237-0300.

Certificate of Mailing/Transmission
(37 C.F.R. 1.8(a))

I hereby certify that, on the date shown below, this correspondence is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the address above on the date indicated below.

November 13, 2003

By: [illegible]
Kevin R. Erdman, Registration No. 33,687
Name of Registered Representative

Respectfully submitted,

[illegible]

Kevin Erdman
Registration No. 33,687
Attorney for Applicant
Baker & Daniels
300 North Meridian Street, Suite 2700
Indianapolis, IN 46204
Telephone: (317) 237-0300
Facsimile: (317) 237-1000

Office de la propriété intellectuelle du Canada

Un organisme d'Industrie Canada

Canadian Intellectual Property Offic

An Agency of Industry Canada

Bureau canadien des brevets

Certification

La présente atteste que les documents ci-joints, dont la liste figure ci-dessous, sont des copies authentiques des documents déposés au Bureau des brevets.

Canadian Patent Office

Certification

This is to certify that the documents attached hereto and identified below are true copies of the documents on file in the Patent Office.

Specification and Drawings, as originally filed, with Application for Patent Serial No: **2,402,761**, on September 10, 2002, by **CONCEPTIS TECHNOLOGIES INC.**, assignee of Thomas Bennett and Sam Guembour, for "Web Engine".

[illegible]

Agent certificateur/Certifying Officer

September 29, 2003

Date

Canada

(CIPO 68)
04-09-02

OPIC CIPO

WEB ENGINE

Field of the invention

The present invention relates to a search engine as more specifically described in the following document and the 4 annexes.

1 INTRODUCTION 4

1.1 Design overview 4

2 CONTENT DELIVERY ENGINE (CDE) 5

2.1 Purpose 5

2.2 Technical Considerations 5

2.3 Architecture - MVC-2 Model 6

2.4 STRUTS implementation 7

2.5 Physical component view 10

3 USER MANAGEMENT SYSTEM (UMS) 10

3.1 Purpose 10

3.2 Terms and Definitions 11

3.3 Profile structure 11

3.4 Groups Structure 12

3.5 Registration template 12

4 CONTENT MANAGEMENT SYSTEM (CMS) 14

4.1 Purpose 14

4.2 Technology 14

4.3 Consideration 14

5 CONCEPTIS ADMINISTRATION LAYER (CAL) 14

5.1 Purpose 14

5.2 Architecture 14

5.3 Functionalities 15

6 DATA WAREHOUSE 15

6.1 Purpose 15

6.2 Architecture 15

6.3 Reporting 15

7 INTERACTION 16

7.1 API's 16

7.2 Rules and Business logic 16

7.3 CMS interaction architecture 16

8 MODULES DEFINITION 17

9 TECHNICAL SPECIFICATIONS 17

Introduction

This section gives an overview of Conceptis Web Engine illustrating the different modules and their interaction. Three key elements comprise the engine: the content delivery engine, the user management system and the content management system. All three elements are explained in more detail.

Design overview

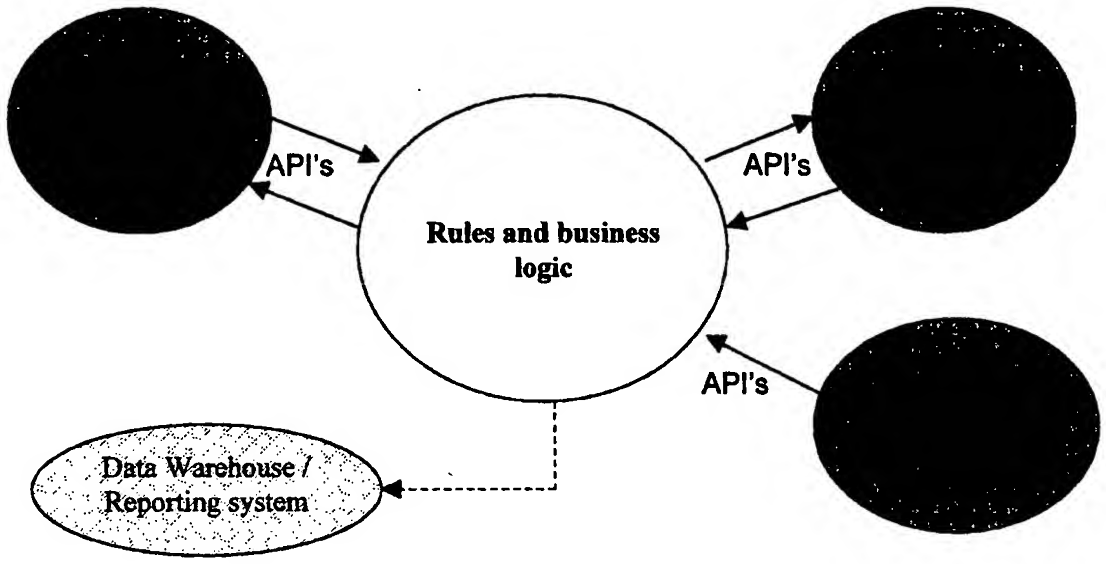


Figure 1.

This design illustrates the key elements of the Conceptis Web Engine interacting with the rules and business logic of any website or portal.

The three main elements are

- Content Delivery Engine (CDE)
- User Management System (UMS)
- Content Management System (CMS)

Another important component of the Conceptis web engine is the Data Warehouse and Reporting system. It is identified differently in the drawing because it doesn't interact with the other components; rather it records all actions of the users.

Content Delivery Engine (CDE)

Purpose

The purpose of the content delivery engine is to pull information from the CMS, respecting the business logic and filters from the user management system, and d liver it as a web site.

Technical considerations

The Conceptis delivery architecture must provide a delivery framework/architecture that allows us to :

- Describe/define a "content view" as a website without the need of advanced programming, e.g. by using design templates.
- Address the evolving nature of the display of content according to a customer's changing needs in a timely manner by providing a mechanism to better define a customer's requirements;
- Associate productivity with efficiency in the creation of websites.

The above considerations can be best addressed using an MVC-2 architecture approach, which decouples the **presentation** aspect of website content delivery from the **business rules** used to access the content, for example from a content management system, and the **control mechanisms** by which the business rules are invoked.

With an MVC-2 architecture the Conceptis delivery framework is able to:

- Use a dispatcher to handle preliminary context setup for incoming website requests, such as the session and user information;
- Use a "Request Container" concept to manage the coordination of website presentation and business logic. The request container may exist either as a servlet, passing control to a JSP template, or can be implemented directly as a JSP;
- Allow the presentation portion to be controlled by JSP templates referencing custom tags that implement the decoration components, buttons, backgrounds, etc.
- Implement the business logic with a combination of custom tags and Java beans or class files.

The above approach ensures that the overall website presentation logic is relegated to the Request Container. Different views of the same website can be obtained by defining various "Request Containers" and associated "Layouts" whil keeping the same content.

Each piece of content itself requires a separate business logic and corresponding presentation logic. We introduce the concept of a vBean and a contentBean to reflect this- the vBean being the presentation portion and the contentBean being the business logic portion.

The de facto standard for such an architecture may be found in the Struts framework, documented at http://jakarta.apache.org/struts/index.html .

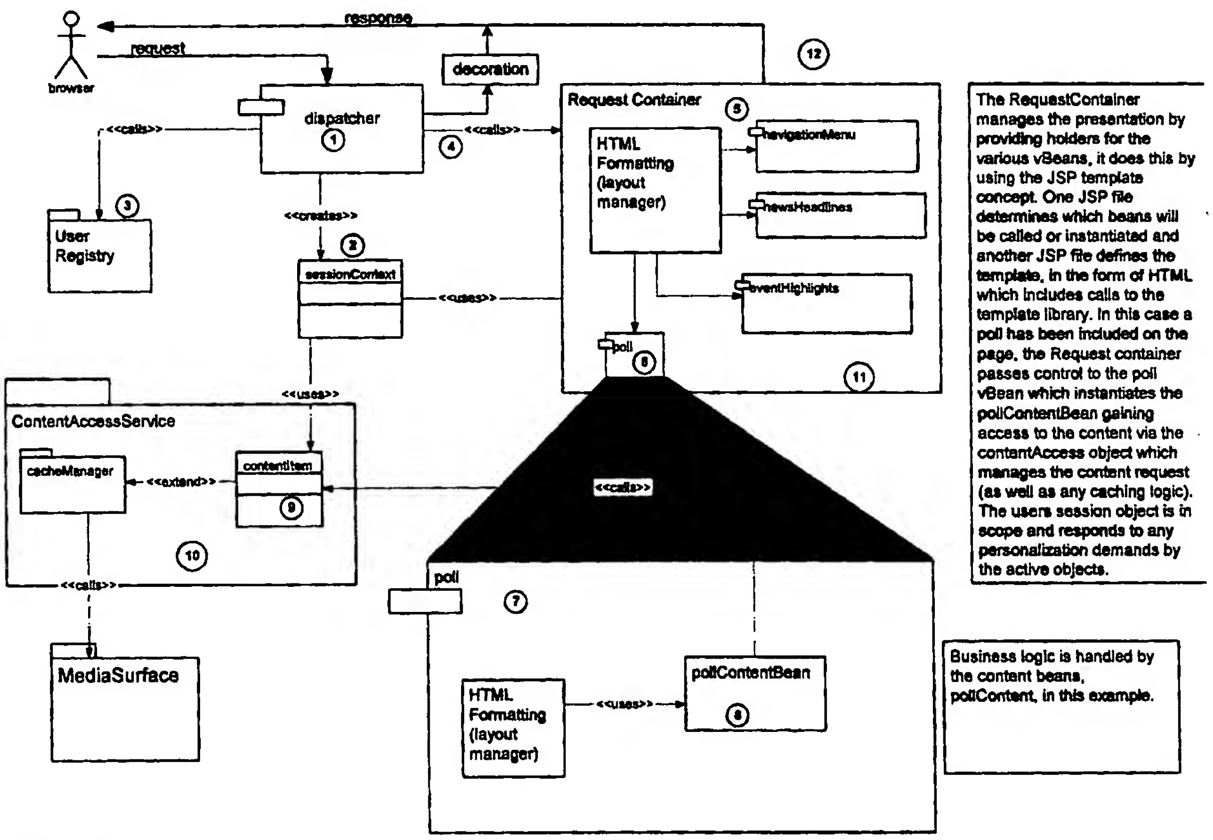
Architecture - MVC-2 Model

The overall delivery flow implemented using an MVC-2 approach is described below (see Figure 2):

1. The **Request Dispatcher** receives all HTTP calls
2. It is the Request Dispatcher's responsibility to ensure a that **sessionContext** object exists for the particular website request and if not, take appropriate action, for example pass control to the "Login" module.
3. The sessionContext consists of all previously defined beans in their previous state ensuring the session's persistence between calls as well as the member's information as obtained via the "**User Registry**" service. The sessionContext also includes a reference to the **ContentAccessService** which will proxy all requests for specific content by subsequent business logic.
4. The Request Dispatcher invokes a url-mapping logic to determine an appropriate RequestContainer to coordinate the actual response back to the user's browser. The Request Dispatcher will also determine whether the incoming request requires a container or not, in the case of requests for "Decorations".
5. The **Request Container** receives control at this point and is responsible for coordinating the invocation of any services or beans. It does this by interpreting two JSP files. The first JSP file functions as a layout manager defining place holders for various component functions, or **vBeans**, such as the navigation menu, banner, poll, etc... The second JSP file includes the actual references to the JSP objects themselves (*The Request Container is an implementation of a **JSP Template***)
6. The Request Container passes control to each of the referenced vBeans.
7. Each vBean receives control and processes the request accordingly. Each vBean implements two main functions: presentation and content. The presentation portion of a vBean is usually implemented in JSP (possibly as a JSP template) and references the associated **contentBean** to get access to the content to be displayed.
8. The contentBean references the **contentItem** in the **ContentAccessService** package.

9. The contentItem itself determines whether to invok further logic or to access the content directly if it finds that the particular content it m already exists.
10. It is the responsibility of the ContentAccessService to provide any caching requirements between the business logic layer and the Content Management System interface (Mediasurface, for example).
11. The Request Container processes all referenced vBeans similarly (6-10)
12. The Request Container returns the response to the requesting browser.

Sample request life cycle



The RequestContainer manages the presentation by providing holders for the various vBeans, it does this by using the JSP template concept. One JSP file determines which beans will be called or instantiated and another JSP file defines the template, in the form of HTML which includes calls to the template library. In this case a poll has been included on the page, the Request container passes control to the poll vBean which instantiates the pollContentBean gaining access to the content via the contentAccess object which manages the content request (as well as any caching logic). The users session object is in scope and responds to any personalization demands by the active objects.

Business logic is handled by the content beans, pollContent, in this example.

Figure 2.

<u>STRUTS implementation</u>

In a Struts implementation the dispatching is handled by the ActionServlet class which can be used, for the most part, as is. The ActionServlet implements a "forward" feature that is configured in the "struts-config.xml" file.

Refer to figure below for the following description of a sample request sequence:

1. The request is deliv red to th Action S rvlet which has read the "struts-config.xml" that defines th URL mappings and their forward actions.
2. The Action Servlet invokes the appropriate Action class for the given request.
3. Each Action class extends org.apache.struts.Action and also the com.conceptis.user.session class
4. The invoked action class ensures that a session object exits or can be created from the requested url and parameters, invokes any generic business logic, stores session beans, and returns control to the Action Servlet.
5. The Action servlet redirects the result from the invoked Action class according to the return result, or "forward", possibly sending the user back, for example, to login.jsp.
6. In the case of a return of "success", the Action servlet passes control to a "Container" JSP which defines what other components need to be invoked or executed.
7. The Container JSP is essentially a JSP template that chooses the appropriate beans to be placed in the Layout Manager, i.e. that the results of home_page.jsp are to be outputted into the MAIN part of the layout Equally it may decide that display_)news_article.JSP will be displayed in MAIN.
8. The Layout Manager is now responsible for assembling the output from the various component invocations by invoking each <template:get ....> according to what the Container JSP has previously decided.
9. Each component invocation essentially is a call to a vBean which may use previously executed business logic. The vBean is now responsible for outputting the result of subsequent contentbeans. For example, in the case of News, the vBean will make a call to generic NewsContent Business Logic (BL), which may use any session objects, e.g. the User object in the case personalization needs. The vBean is responsible for assembling the output from the business logic.
10. The process continues for each component of the Container until the response to the user has been completely assembled.

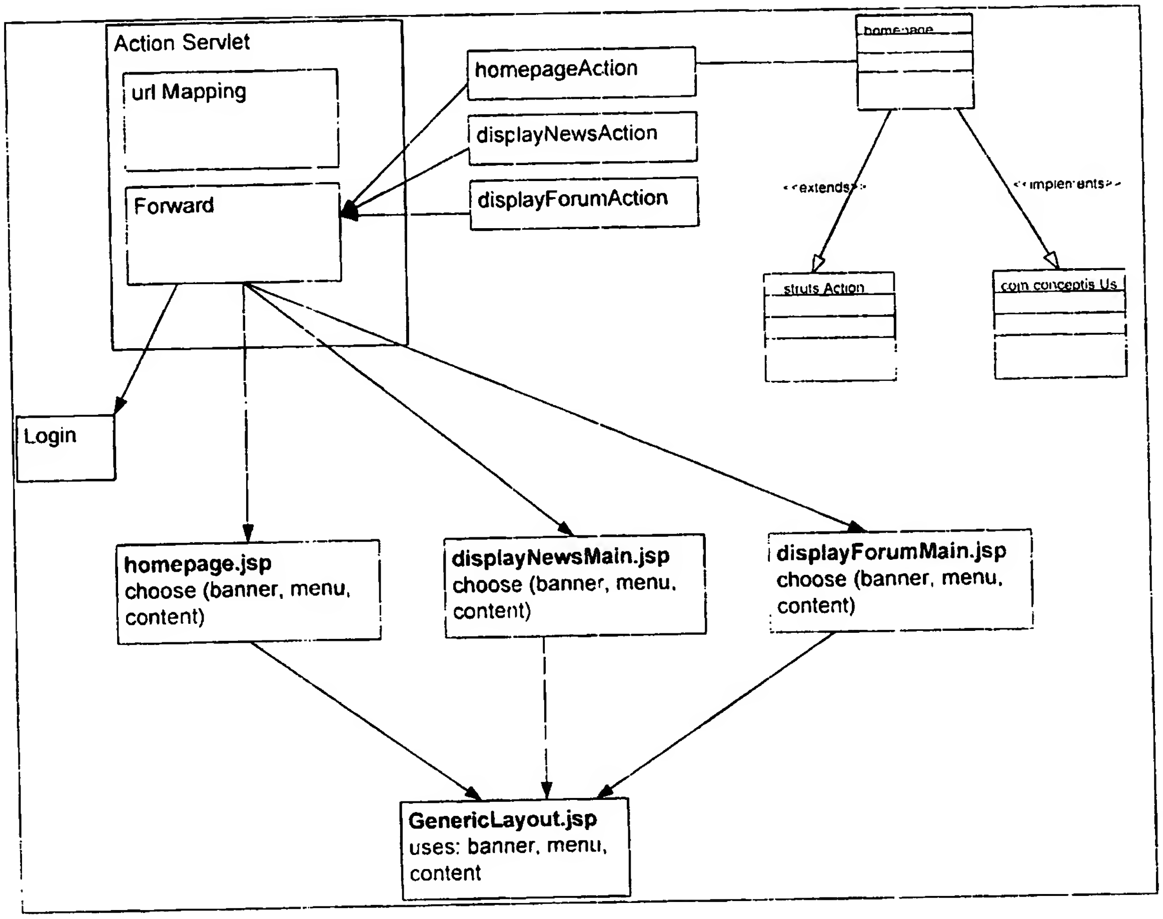

Action Servlet
url Mapping
Forward
homepageAction
displayNewsAction
displayForumAction
homepage
<<extends>>
<<implements>>
struts Action
com conceptis Us
Login
homepage.jsp
choose (banner, menu, content)
displayNewsMain.jsp
choose (banner, menu, content)
displayForumMain.jsp
choose (banner, menu, content)
GenericLayout.jsp
uses: banner, menu, content

Figure 3.

Physical comp nent view

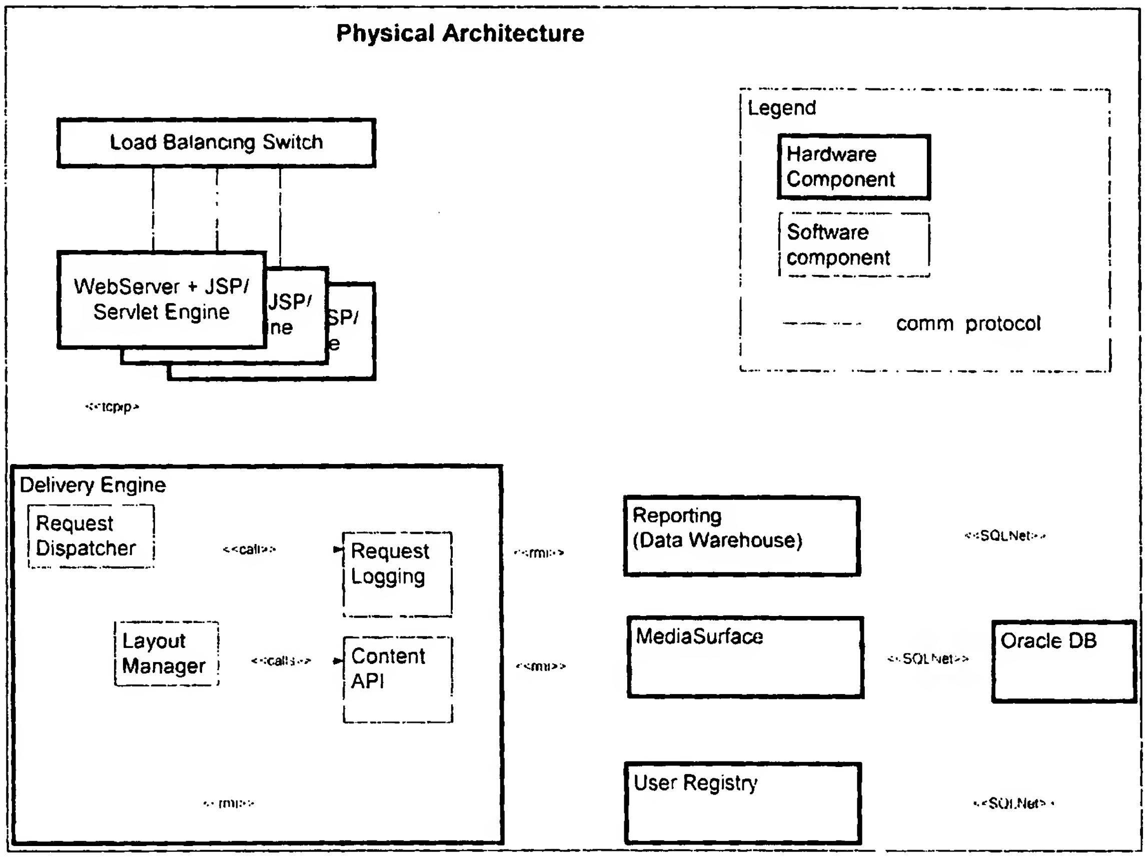


Figure 4.

User Management System (UMS)

Purpose

The purpose of the User Management System is to easily manage access control, classification, preferences, and personalization of the content delivered to users. It is closely linked to the registration process. In effect, it applies filters to what content is available to each user.

The user management system also provides the ability to:
- generate accurate reporting
- track user attributes
- manage access control
- track users' participation in promotions and contests

Terms and d finitions

User attributes have many different uses, and we need a clear set of terms to define these. Here's how those terms work from a user management point of view.

Access control is a security-related concept. We want to prevent most users from accessing the administrative features of our sites, such as content management or forum moderation. We often do this by specifying roles and permissions.

Classification covers a wide range of attributes, most of which apply to the user as a person, regardless of the site: medical specialty, profession, interests, and so on. We use these attributes during content delivery to determine which pieces of content (or ads) would be most interesting to the user and to promote that content by highlighting it or by hiding irrelevant content.

We also use those attributes when generating stats. It is often useful to know, for instance, how many of our users are fully accredited doctors, or what our most frequent visitors have in common.

Preferences give users the ability to directly influence how a given site is presented to them. We could let each user decide which elements (headlines, calendar, poll, etc) should appear on his welcome page. We should let users decide how much information they want us to mail them (newsletters, ads, etc). Classification influences site behavior based on who the user is; preferences influence site behavior based on what the user asked for.

Personalization is mostly a content delivery term. It is the process of tailoring website content to the user. It makes use of both classification and preferences.

Profile structure

The user management system uses multi-level profiles to provide us with sufficient flexibility.

The top-level profile consists of personal data. This covers information that defines the user independently of our sites. As much as possible, a person should only have one personal profile, no matter how many sites that person may visit.

Then comes the user profile, where we give the person a username and password. The user profile is not tied to a specific site. Rather, it represents the user's account in the Baxter framework as a whole. We will probably encourage people to create and use only one user profile, but it is technically possible for a person to have multiple user profiles.

The next level is the site profile. It defines a user's rights and preferences in relation to a specific site. A user profile can have any number of site profiles. For more flexibility in designing our sites and access levels, we could subdivide some sites into a hierarchy of subsites. A user could then have a site profile for each subsite. The user's permissions could be determined by reading the lowest-level site profile or by combining all site profiles in the current branch.

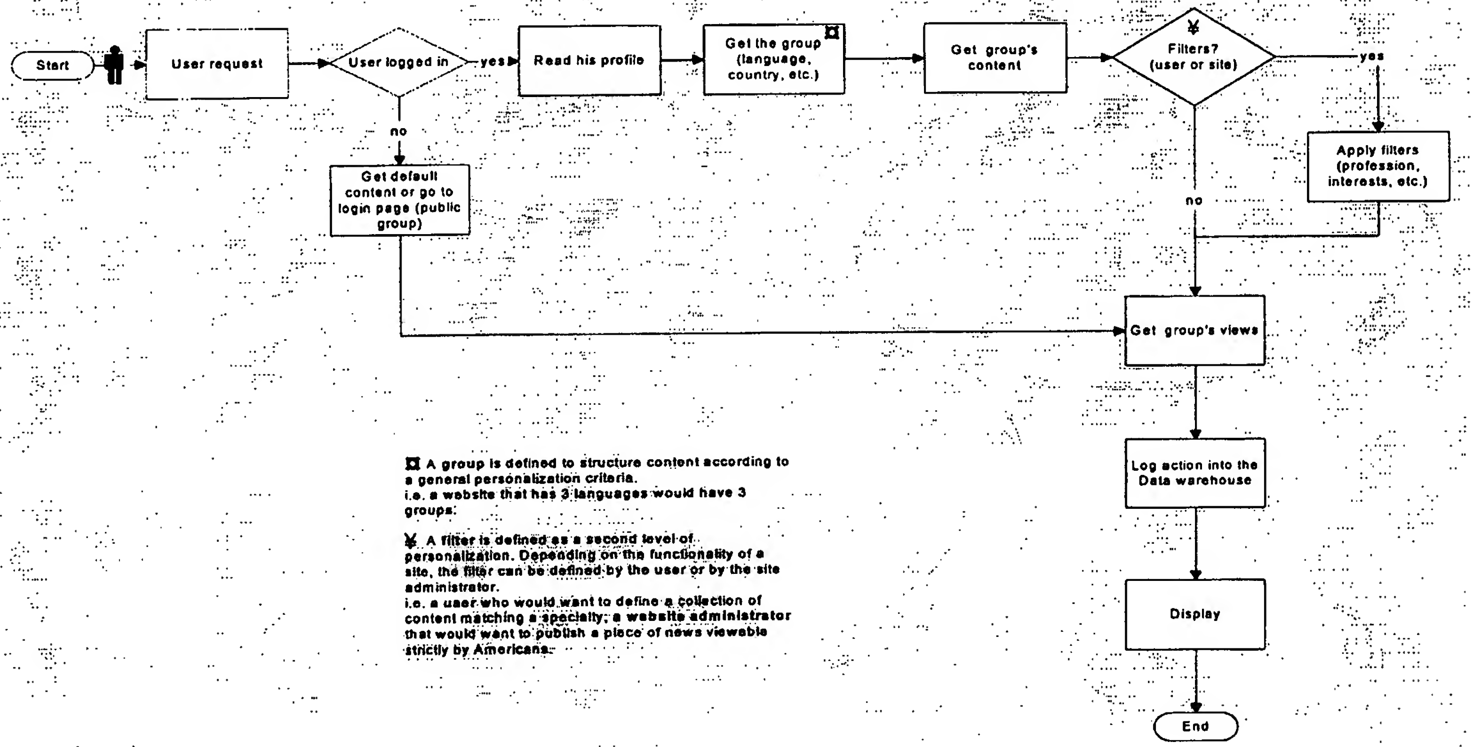
Group structure

The user management system also provides the ability to structure content according to general personalization criteria. Multi-language sites are good examples of how groups can be used. A website that has multiple languages would have the same number of groups as it has languages.

R_gistration template

The registration template is the foundation of the UMS. Appropriate definition of the fields that compose the template is crucial. Users see the registration template as a web page, but its content defines how content will be delivered to groups of users and/or to individuals.

Personalization flow - e


Start
User request
User logged in
yes
no
Read his profile
Get the group ¤ (language, country, etc.)
Get group's content
¥ Filters? (user or site)
yes
Apply filters (profession, interests, etc.)
no
Get default content or go to login page (public group)
Get group's views
Log action into the Data warehouse
Display
End

¤ A group is defined to structure content according to a general personalization criteria.
i.e. a website that has 3 languages would have 3 groups.

¥ A filter is defined as a second level of personalization. Depending on the functionality of a site, the filter can be defined by the user or by the site administrator.
i.e. a user who would want to define a collection of content matching a specialty; a website administrator that would want to publish a piece of news viewable strictly by Americans.

Figure 5.

Content Management System (CMS)

Purpose

The content management system has extensive facilities for managing of the flow of content, the versioning of documents, and the mapping of relations between content. All content — articles, news, images, font types, etc — is stored in the same database. Before storage, a classification of the document is done to assign it different properties. It is then assigned to a precise position or positions in the repository, just as files are assigned to folders in Windows.

T chnology

Conceptis has chosen Mediasurface as its Content Management System. Complete documentation of this product can be found at www.mediasurface.com.

Consideration

Other technologies (e.g. Interwoven) could interact with both the Content Delivery Engine and User Management systems. Our current development is aimed at Mediasurface, but other technologies are being considered for future development.

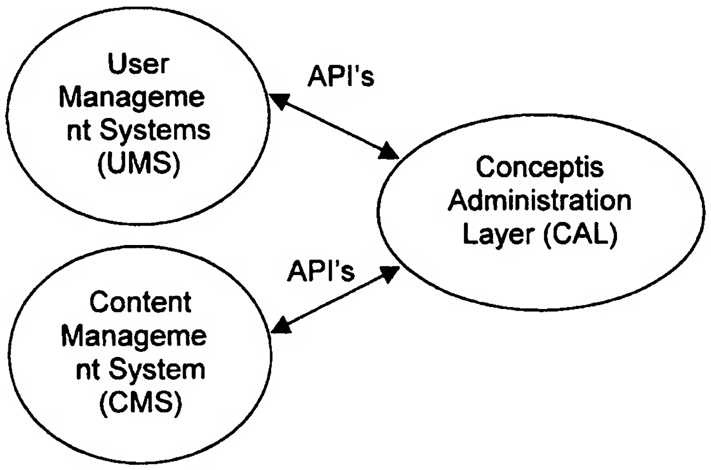
Conceptis Administration Layer (CAL)

Purpose

The Conceptis administration layer was developed so content users can enter and manage content in the CMS. This web interface was developed to answer the needs of editors, writers, integrators, and content administrators.

Architecture

The CAL interface serves as a bridge between the UMS and CMS. When editors place content in the CMS, they give that content the necessary attributes for it to be handled properly by the business logic and rules of the target website or other delivery medium.


User Manageme nt Systems (UMS)
API's
Conceptis Administration Layer (CAL)
API's
Content Manageme nt System (CMS)

Figur 6.
Functionalities

The CAL permits content users to:

- Manage access control to content
- Upload content items
- Create, edit and view items
- Index content
- Search for specific content
- Find and replace
- Remove content, either individually or by batch
- Generate reports

Data warehouse

Purpose

In order to provide accurate statistics about site use, Conceptis has developed a Data warehouse on an Oracle database. All actions on a site are recorded in logs that are copied every night to the database, giving us the needed information for the production of statistical reports. This module is complementary to the Conceptis Web Engine but provides considerable value-added to the product, for obvious marketing reasons.

Architecture

Conceptis uses the snowflake model with processed cubes, a set of tables comprised of a single, central fact table surrounded by normalized dimensions. Note that a "cube" is a diminutive name given to a fact table. A processed cube refers to a feature of OLAP technology that enables a reporting engine to look at slices of data very efficiently in outputting multi-level reports.

Once a day an OLAP filter retrieves the hits of the day and processes them into our snowflake schema database, also called our main cube. The OLAP filter also creates and populates the aggregate data cubes. An aggregate data cube is a slice of data taken from the main snowflake database and is used to accelerate a specific type of report.

R porting

To create reports, we use the reporting software package from Crystal Reports. Crystal Reports is OLAP-compatible, inexpensive, and very standard in the industry.

Interaction

The Content Delivery Engine, the User Management System, and the Content Management System all interact together to deliver websites. This section illustrates how this interaction is done and the technology behind it.

API's

APIs are a set of JSP tags and related Java class files that provide high-level access to the Conceptis framework, including the CDE, UMS, and CMS.
If we refer to Figure 1, we can see that the different modules interact together via different APIs that all respect the same business logic. These APIs can invoke the available functionalities required to produce a website. In the Conceptis web engine the different content items and functionalities are accessed thru Java tags and libraries.

Rules and business logic

Again on Figure 1, what we have called the rules and business logic specifies what is made available to users in the form of a website. Every new website has its own set of rules and business logic independent of content and users. Each website employs a series of calls to the APIs using Custom Tag libraries, producing all the required views.

CMS interaction architecture

Conceptis has developed two different APIs to interface eventually with any CMS. The first API is a generic content API. This API delivers and pulls content in a generic way from the CMS. The second is a vendor-specific driver API. This one serves as a translator between the generic API and the proprietary CMS.

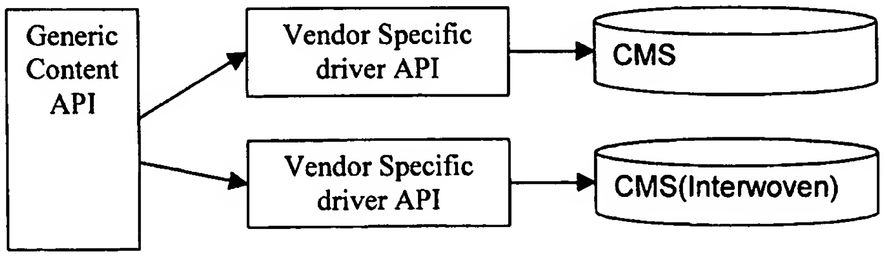


Figure 7.

Module definition

Conceptis has developed numerous modules with their own functionalities, all of which are sharable among websites and are independent from content. Below ar a list and a short description of the modules that Baxter has expressed interest in.

Forum

A forum is composed of any number of topics under which can be posted any number of messages. There is no separate administration interface; rather, the administrative options are integrated into the forum display and made visible only to authorized persons.

Poll question

A poll question is normally available for users to answer. These questions either can be of the opinion type with no correct response or can have a specific correct or incorrect response. The answers can be true/false, yes/no, or multiple-choice. Poll questions may or may not be linked to other content.

CEU/CME tracker

This feature allows a user to maintain a profile to track his or her CME/CEU on all websites run from the same platform.

Other available modules

- Messaging center
- Calendar
- To-do-list
- Groups
- Virtual registry
- File cabinet
- Survey module
- News feed
- Comment content
- Etc

Technical Specifications

To run the Conceptis Web Engine, the following hardware and software systems are required as minimal requirements.

Web server

- Sun Single UltraSPARC II 500 MHz processor or higher
- 256 Mb RAM
- 2 Gig hard drive space
- 10/100 NIC
- Solaris 8 OS

Database server

- Sun Dual UltraSPARC II 500 MHz processor or higher
- 512 Mb RAM
- 4 Gig hard drive space
- 10/100 NIC
- Solaris 8 OS
- Oracle 9i

Networking

- 3 megabit dedicated line to the Internet

Content management software

- Mediasurface 4.05

Reporting software

- Crystal Report Enterprise

ANNEX 1

USER MANAGEMENT

USER OBJECTS IN THE NEW FRAMEWORK

by Eric Gauthier

Document description

This document explores the many issues related to user management and attempts to define a new user management engine that could be used by all Conceptis applications.

Table of content

1. INTRODUCTION 23
1.1 Revision history 23
1.2 Terminology 23
1.3 References 24
1.4 Requirements 24
1.5 Contacts 24
1.6 System description 24
2. NEEDS 25
2.1 Conceptis 25
2.2 Clients 26
2.3 Users 26
3. CONCEPTS INVOLVED 27
3.1 Unique identification 27
3.2 Access control, classification, preferences and personalization 27
3.3 Context-dependent attributes 28
3.4 Reference lists 28
3.5 User validation 28
3.6 Profile completion 29
4. WHAT'S BEEN DONE 30
4.1 theheart.org 30
4.2 VOICE/360 31
4.3 CTC 32

4.4 KOL 33

5. PROFILE STRUCTURE 35

5.1 Site families 35

5.2 Classification 35

5.3 Custom attributes 37

5.4 Access control 38

5.5 Preferences 38

5.6 Site communications 38

5.7 User relationships 39

5.8 Admin 39

5.9 Stats 40

6. COMMON DATABASE 41

6.1 Accessing the data 41

7. MEDIASURFACE 42

7.1 User mapping 42

7.2 Reference lists 42

7.3 Persons 44

8. USE CASES 45

8.1 Regular user experience 45

8.2 User management 50

9. BASIC PROFILE 51

9.1 Person attributes 51

9.2 User attributes 52

10. PRIVACY ISSUES 54

10.1 Protection of personal information 54

10.2 Usage stats 55

11. INTERNATIONALIZATION 57

12. ACCOUNT MATCHING 58

12.1 Basic matching technique 58

12.2 Matching within sites 58

12.3 Matching across sites 59

13. STATS 60

13.1 The distinction between stats and operations 60

13.2 The data warehouse 60

14. USES FOR THE PERSON OBJECT 61

14.1 Contact info 61

14.2 Authorship 61

15. MODULES AND INTERFACES 62

15.1 Site-based operations 62

15.2 User management module 62

15.3 Common database 63

16. QUESTIONS 64

INTRODUCTION

<Explain here what is the goal of the project or sub-project>

Revision history

Revision	Author	Description of Changes
June 7, 2002	Eric Gauthier	

Terminology

user	Any person that accesses and uses our systems (either internal systems, like content management interfaces, or public systems such as websites).
user profile	A data structure containing everything we know about a user. A person may have more than one user profile, although this should generally be avoided.
user object	A programming object representing a user within our systems.
attribute	Generally, a property of an object. In this document, a property of a user (name, date of birth, etc.)
client	A company or other entity for which we create a site. The client makes decisions regarding what should appear on the site, what kind of users should be validated, and so on. For some sites (theheart.org, jointandbone.org), Conceptis is the client.
context	An environment in which a user object is used. It can be a site, a subsite or an independent application.
CTC	Clinical Trial Central. A highly secure product with multi-site capabilities, used by medical companies to relate to their trial investigators.
KOL	Key Opinion Leader. A product with multi-site capabilities, used by medical companies to perform market research by interacting with doctors. More standardized and adaptable than CTCs.
access control	The process of specifying which users are allowed to perform which actions and see what content, and of enforcing those specifications. This is often crucial, as the content may be highly confidential.
classification	The process of associating certain properties with users, which allows for personalization of content (upon delivery) and for statistical analysis based on those properties.
personalization	The process of adapting the display and behaviour of an application to fit a specific user's needs and

	tastes.
preferences	A set of choices made by one user which determine the behaviour of certain applications.
role	A role is a way in which a user relates to a site (e.g.: user manager, editor, etc). A user's roles define what the user is allowed to do on that site (usually, through a set of permissions)
permission	The right to perform a certain action or type of action.

References

[1] The Platform for Privacy Preferences (P3P) Project (http://www.w3.org/P3P/). An initiative of the World Wide Web Consortium (W3C) to define a standard for handling personal information. It seems to focus on allowing users to control and be aware of how their personal information is collected and used.

[2] The Personalization Consortium (http://www.personalization.org/). I haven't read the site yet, but this group seems to address a lot of the issues we'll be facing.

[3] Everything You Need to Know About Personalization (http://www.wdvl.com/Authoring/ASP/Personalization/index.html). An article on personalization.

Requirements

[1]

Contacts

Sam Guembour	Architect
Eric Hechinger	Technology lead

System description

This system should fulfill all our user management needs. It should not be site-specific.

Needs

Conceptis

In general

Conceptis users in general will need:

- consistency across sites and products, both at the data model level and at the interface level

Site administration

For administrative purposes, we need:

- the ability to validate potential users based on site-specific criteria
- a user-friendly set of interfaces to dig into the user repository and perform various operations (creating and deleting users, unlocking them, finding multiple accounts used by a single person)
- standardized user profile import procedures

Content

Our Content department will need:

- the ability to track copyright information by designating users as authors or copyright owners

Marketing

Our Marketing department will need:

- the ability to generate mailing lists and do proper testing before each mass mailing
- the ability to track users' mailing preferences in order to respect spam laws and provide the best possible communication with users

Stats

The people responsible for generating or designing our stats reports will need:

- a well-defined set of standard user attributes
- coherent use of constants in our data model
- the ability to follow a user across sites (to confirm, for instance, that a page view on site B is the consequence of a banner click on site A)
- the ability to clearly identify Conceptis employees, employees of our client companies, and other users who should not always be included in our stats reports

Security

For security purposes, we will need:

- coherent, hack-proof login processes including various standard tools: account lock-out, forced password changes
- a flexible, well-documented password policy

Programming

The Programming departement will need:

- re-usable code
- the ability to easily handle user movement between sites

Clients

Cli nts who buy or use our various products will need:

- statistically accurate reporting
- the ability to track user attributes that are important to them
- solid access control, to ensure only the right people view their data (this ties into content management, of course)
- the possibility of tracking users' participation in promotions, contests and so on

Users

End-users on any of our sites will need:

- intuitive login interfaces
- simple transition processes between sites (where applicable)
- respect of their privacy (and confidence in that respect)
- easy access to the part of their profile they're allowed to modify
- clearly-documented password retrieval techniques (where applicable)
- clear, consistent site personalization techniques

Concepts inv lved

Unique identification

Unique identification implies a one-to-one correspondence between persons and user profiles. Our current sites tend to be independent from one another, so that a person registered on three sites will have three separate user profiles. Furthermore, it is even possible (though uncommon) for a single person to have multiple profiles within one site.
It is to our advantage to give each user a single profile that will be used across sites. This would make it easy for us to let users move between sites (to have jointandbone users access CyberSessions on theheart, for instance), and would allow us to generate interesting usage stats for our own purposes.
However, this may be construed by some users as a privacy breach. Decisions about this should not be taken lightly.
Note: on the Merck CTC, we're exploring the possiblity of using the American M dical Association's Verisign ID program for medical professionals. This program would essentially give each doctor a unique ID. If the program catches on, we may find ourselves using those IDs on other sites. These could help us match and combine a user's profiles from various sites.

Access control, classification, preferences and personalization

The terms

User attributes have many different uses, and we need a clear set of terms to define these. Here's how those terms work from a user management point of view.

Access control is a security-related concepts. We want to prevent most users from accessing the administrative features of our sites, such as content management or forum moderation. We will often do this by specifying roles and permissions.

Classification covers a wide range of attributes, most of which apply to the user as a person, regardless of the site: medical specialty, profession, interests and so on. We use these attributes during content delivery, to determine which pieces of content (or ads) would be most interesting to the user, and to promote that content somehow (by highlighting it, or by hiding irrelevant content). We also use those attributes when generating stats: it's often useful to know, for instance, how many of our users are fully accredited doctors, or what our most frequent visitors have in common.

Preferences give users the ability to directly influence how a given site is presented to them. We could let each user decide which elements (headlines, calendar, poll, etc.) should appear on his welcome page. We should let users decide how much information they want us to mail them (newsletters, ads, etc). Classification influences site behaviour based on who the user is; preferences influence sit behaviour based on what the user asked for.

Personalization is mostly a content delivery term. It's the process of tailoring website content to the user. It makes use of both classification and preferences.

The tricky part

The boundary between access control and classification is a fuzzy one. Access control is mostly about site *functionality*, while classification is mostly about site *content*. However, there may be some cases (on CTCs, especially) where we want to prevent users from viewing a piece of content because that content is confidential. This should probably qualify as access control. Our access control techniques, then, must allow for a few different types of criteria.

Context-dependent attributes

In most of our systems, we've been dividing user profiles in two parts: a generic profile and one or more site-specific profiles. The underlying idea is that some of a user's attributes may change depending on the context.
A user's name will always remain the same, no matter what the user is doing or what site the user is consulting. The user may want to change his or her name at some point, but that change will apply to all possible contexts. On the other hand, a user may be considered an editor on one website, but only a regular user on another site. The user's role attribute is context-dependent.

Reference lists

Many user attributes will have a fixed set of "legal" values. We enforce these to nsure better coherence in our stats and in our access control. When filling in their profile, users generally aren't allowed to type their gender, or the name of their country; they must pick from a list.
So far, such lists tend to be redefined for each site, which makes it harder to maintain those lists or combine stats from various sites. We should consider creating one set of canonical reference lists that would be used by all our products.

[redacted]

User validation

On some of our sites (the ones we build for various client companies), people aren't allowed to register by themselves. Instead, if they want a user account, they have to send their profile to the client company, which validates it and then sends it to us.
Many of our sites, though, have a registration form that users can fill in. What happens after we get the form data varies from site to site. On some sites, a user account may be created immediately, allowing the new user to enter the site right away. On some other sites, we may have to confirm that the candidate user is an accredited doctor or an employee of a media outlet. This requires manual validation, although we sometimes allow for shortcuts: if the person has an email address from a hospital, for instance, we may validate the account automatically.

At the least, we need to make it easy for our site managers to perform the nec ssary validation. At best, we could standardize our validation techniques across sites and automate them as much as possible.

Profile completion

The more we know about our users, the more we can generate useful statistics (there's privacy issues, of course; those are covered in section 0).
Users, though, tend to follow the path of least resistance; when entering their profile, most will only fill in the mandatory fields and nothing else. We need to establish some techniques to:

- define which fields are mandatory for each site, and whether some of them may become optional for certain types of users
- keep a library of standard validation techniques (for dates, email fields, etc.)
- if possible, come up with a definition system to store and handle the peculiarities of each site automatically, instead of hand-coding each registration form

What's been done

theheart.org

theheart.org is one of our oldest, most successful sites. It has many flaws, most of them well-hidden. This is a site, after all, that was developed when the company was much smaller and didn't have a database administrator. The site has received many additions and undergone a few restructurations since.

Schema

During its evolution, theheart adopted some multi-site techniques. One of them is th use of a short site code, or schema identifier. Each user account is tied to a specific schema. Access levels are also defined on a schema-by-schema basis.

Classification and stats info

User profiles on theheart include the following information:

- profession
- medical specialty
- interests
- journals read
- other publications read
- associations
- how the user heard of theheart.org

Some of these could be handled with a group/category system such as the one used by the KOL engine (see 0). Others could be stored in an XML field as extra information.

Incomplete profile management

Upon entering their username and password, users may be prompted to complete their profile. A table named user_incomplete stores the message to be shown to each user.

While this is helpful in ensuring we get all necessary data, it could be better executed. The messages could be stored in their own reference table, for instance.

Duplicate account handling

There's a procedure in place to prevent users from having multiple accounts. It goes something like this:

- a user unsuccessfully tries to log in, then opts for the "send me my password option".
- the system reads the user's profile and sends the user's password to the user's email address
- the system checks for similar profiles in the database

- if any duplicates are found, the system sends an email to a Conceptis address listing all the duplicate accounts and pointing out which one is currently in use
- a Conceptis employee manually deactivates the extra accounts

Access levels

The access levels system on theheart essentially defines a user's permissions on a given site. It's a flawed system which combines many concepts: an access level may represent a role, a very specific permission, or a client-defined category. There's even a few levels in there named after specific users.

Each access levels is represented by one bit. We define a user's access rights by combining all the user's access level bits into a single integer. Access criteria for each page are handled the same way. To determine if a user can access a page, we do a bitwise AND between the user's and the page's levels. A non-zero result means access is granted.

This makes for fast comparisons, but there's an obvious drawback: in the current data model, access level bitmasks are restricted to 15 digits, which means a maximum of approximately 48 different access levels.

VOICE/360

There's a precedent to the current re-engineering project. A while ago, we started a project called VOICE. It was meant to define a flexible infrastructure that could run all of our sites and simplify interaction between those sites, thus making it easier for us to build online communities. Its main designers were Marc Bélanger, Michel Émond and Benoît Isabelle. The VOICE project was eventually abandoned, mostly due to lack of resources, but it did at least generate a data model that incorporated multilingual support, highly flexible site structures, roles and permissions, and more.
One site was built using some of the VOICE principles: Pfizer 360. The 360 engine is only a partial implementation of VOICE, but it's handy as a concrete example of the VOICE design.

Persons

Of all our systems, the 360 engine may be the one that makes the clearest disctinction between a user's personal and site-specific attributes.
In most of our systems, user information is split into a user_info table (which holds the username, password and contact information) and a user_sites table (which holds a profile for each site, including permissions in some cases).
The 360 data model, instead, uses a person object as the foundation for each user profile. Person objects hold contact information and any other info specific to the person, but no username, password or anything related to our systems. The person table in the database can thus be used to keep track of people in general, even those who will nev r become users: copyright holders, contacts at various companies, etc.

Any number of user objects can then be associated with a person, each having its own username and password.

Roles and permissions

The 360 engine uses a few different tools for access control. One of these is a set of roles and permissions. A permission gives the right to do a specific action or type of action: view documents, manage user accounts, etc. A role is a group of permissions. A user can have many roles.

Locations

While earlier systems would store user's addresses solely as strings, or make use of separate state and country tables, the 360 data model uses a hierarchical location table. Such a table allows for any number of levels, making it possible to store countries, states and cities in exactly the same way, and allowing the definition of groups such as continents, or regions within a country.

Each user profile has a primary location attribute pointing to the lowest-level geographical data we have for that user (usually a city). We can retrieve a user's state or country by going up the location tree from that low-level location.

CTC

This document will only cover second-generation CTCs (namely, MerckIMPACT.com and STICHtrial.org) since they provide all the features of previous CTCs and are more coherent.
For more details, consult the ANA001-CTC_template_FR.doc document in our SourceSafe repository, under CTC/Analyses, which examines the first generation of CTCs and sets the stage for the second generation CTCs.

Location caching

The current CTC data model includes a hierarchical location table (see 0 above) with four levels (more levels can be added without changing the model):

- Earth
- countries
- states
- cities

In addition to a primary, low-level location attribute, CTC user profiles also have two "cached" attributes. For convenience and speed, we added state and country fields which also use the location table. This allows us to quickly list users by state or country without having to walk up the tree structure for each user's location.

Geographical access control

We use the location table for user's addresses, but also for access control. We can associate a list of countries to each user and to each page. When a user tries to access a page, we compare that user's country list with the country list for the page. If one country matches, we grant access to the page. As a shortcut, we can associate the Earth location to a page; this makes the page accessible everywhere.

Access levels

The access levels system on CTCs is similar to the one on theheart (see 0), although the levels are defined more coherently. The Merck CTC, for instance, defines the following access levels:

- everybody (the most basic level, which signifies access to the site and nothing more)
- role-based levels (content manager, events manager, etc.), each of which gives access to a specific admin interface and grants some related permissions (content managers, for instance, are allowed to see all content, no matter the access criteria).
- sub-protocols (FUTURE1-011 US, FUTURE1-011 ex-US, FUTURE-2 015 Can). These were defined by Merck and simultaneously define medical protocols and geographical regions. For that reason, we do not use the geographical access control system on the Merck CTC: we make every page visible to the entire planet and let the protocols define regions.

Windows-based authentication

Security on the CTCs works at two levels. Each user has an account defined in our database, as well as a Windows NT account on the CTC server. Each of the secure folders on the CTC is configured in the IIS Web server software to prevent anonymous access and enforce a Windows account login.

As a result, the Windows authentication is done before any of our verification code gets executed. For added security, we configure Windows to lock an account after three consecutive login attempts.

This has the advantage of providing a security layer that cannot be affected by any bugs in our programming. The drawback of this system is that we cannot change the look of the Windows login pop-up window, nor interact with the authentication process in any way. Additionally, we recently found the security verification to be imperfect: a user can log in properly, then end his session, then lock his account by entering the wrong password, then attempt another login and enter the site by typing in the right password, even though the account is now locked. It seems the server may remember validated users for a short while (an hour? a day?) after their session has ended, and be more permissive with those users. This is not a major security breach, but it does make the system look very unreliable.

<u>KOL</u>

For more details, see the kol_user_access.doc document in our SourceSafe repository, under KOL/Analyses.

Custom attributes

Since KOLs are meant as market research tools, our clients may want stats based on very specific concepts. They may want to qualify and classify users in ways that we cannot predict. For that reason, we designed the KOL user profiles with flexibility in mind.

The KOL database model includes a set of tables that we can use to define and fill custom user attributes absent from the basic user profile. These attributes can be of three types: date, string or number. Giving each attribute a type makes it possible for us to sort user records based on those attributes, or to implement selection criteria using "greater than" or "less than" operators.

On-the-fly groups and categories

We implement classification and access control in our various systems using many different user attributes: role, medical specialty, interests, affiliations, and so on. In the KOL data model, these are all represented as groups.

Each different classification concept (medical specialty, leadership level, etc.) is represented by a category of groups. Each category has a type attribute indicating what the category should be used for: access control, classification, mailing, and anything else we may decide on. Each category contains at least one group called the default group. Default groups always contain every single user.

Criteria matching for a content object (a page, let's say) is done as follows:

- an admin user selects any number of groups in the categories used for page access, then publishes the page. In categories that hold no selected groups, the system automatically picks the default group, meaning that everybody will meet the criterion for this category.
- when a user tries to access the page, the system compares the user's groups to the page's groups, category by category.
- if there's at least one match in every category, the user is granted access to the page

Profile structure

The new user management system should use multi-level profiles to provide us with the necessary flexibility.
Top level is the person data. This covers information that defines the user independently of our sites. As much as possible, a person should only have one person profile, no matter how many sites that person may visit.
Then comes the user profile. This is where we give the person a username and password. The user profile is not tied to a specific site. Rather, it represents the user's account in the Conceptis framework as a whole. We will probably encourage people to create and use only one user profile, but it will be technically possible for a person to have multiple user profiles (practical for people who don't like putting all their eggs in a single basket).
The next level is the site profile. It defines a user's rights and preferences in relation to a specific site. A user profile can have any number of site profiles. For more flexibility in designing our sites and access levels, we could subdivide some sites into a hierarchy of subsites. A user could then have a site profile for each subsite. The user's permissions could be determined by simply reading the lowest-level site profile, or by combining all site profiles in the current branch (more on this later).

Site families

In order for the new system to make sense and keep its current security level, we will need to group our sites into families.
In most cases, we'll create families based on site ownership, since privacy policies and security concerns are usually tied to the owner of a site.
Each site family would essentially have its own pool of person and user profiles. A person could have multiple user accounts if the site family has multiple levels of security. For instance, we could create a Duke family made up of STICHtrial.org and the DUCCS KOL. Since STICHtrial is a CTC and has a higher security level, it could have a set of users distinct from the KOL users, although both sets would share the same person profiles.
We'll also need a common database that can be accessed by any family. We'll us it to store lists of sites and so on. For more details, see section 0

Classification

At which level should classification be done? It's rather pointless at the site level: an interventional cardiology specialist is an interventional cardiology specialist no matter which site he or she is visiting. If we want to generate interesting cross-site statistics, we need to place most user attributes at a higher level.
Even categories that seem site-specific may be better at the user level, since many sites in the same family may share the same preoccupations. Categories defined for a specific site and client (based on medical protocols, for instance) may be re-used if we create a sibling site for the same client within the same family.

Do we then apply classification techniques at th user or the person level? Here are the advantag s to ach approach:

p rson level:

- easy to do statistical breakdowns in cross-site usage reports
- user experience is more consistant within one site family
- users can create multiple accounts without having to re-enter this data every time

user level:

- gives users a little more flexibility in defining their accounts within one site family
- helps define the difference between persons and users: a person exists independantly of our systems and thus is not classified according to our systems' categories

We could solve this by forcing people to have exactly one person profile and one user profile, and attaching classification to the user profile.

Group system

Here's what we can use for maximum flexibility. Keep in mind this is an ideal structure, but:
1) it may be too slow to use, in practice
2) it can be a lot of work to synchronize this with MediaSurface. When we first put MediaSurface to use, we'll have to use a different technique and slowly phase into this model. See section 0 for details.
We use groups and categories to implement our reference tables and handle all of our classification criteria. If we need to record users' medical specialties, for instance, we create a category called "medical specialty". Within that category, we create one group named after each specialty. All sites will use this list of specialties; in order to simplify user interfaces, though, some sites may define a mask so that only a subset of the medical specialty groups will be considered.
If we allow for parent-child relationships between groups, we can define reference trees. This allows us to define sub-specialties, for instance, or to create a single tree of geographical locations, from the Earth (as the root) down to cities or even streets.
If we do use hierarchical relationships between groups, each category could be made into a proper tree of groups, with the root of the tree being an "everybody" group.
Additionally, each category would need a set of level definitions, so that we can easily differentiate cities from countries, for instance.
Category definitions would be as follows:

- category ID
- category name
- description
- display order

Group definitions would be as follows:

- group ID
- group name
- abbreviated group name
- description

- creation query (SQL; optional)
- display order
- availability (public or private)
- manager/owner
- confirmation flag
- level
- ID of parent (if any)
- ID of category

[redacted]

Using groups as reference lists

By defining groups and group categories in the common database and exporting those definitions to each site family, we can greatly simplify cross-site and cross-family stats, *and* reduce the amount of data maintenance needed.

To help us expand those reference lists and to accommodate all those "other (please specify)" fields in our registration forms, we can give each group a confirmation flag. Users can then enter "other" values that are not in our lists, and for each such new value we can create a new group and flag it as being unconfirmed. If many users enter the same value (with the same spelling), they will all be added to that unconfirmed group. Periodically, we can sift through the unconfirmed groups and confirm the most popular ones (this will often require us to merge groups representing alternate or incorrect spellings).

As if this wasn't messy enough, here's a possible complication. The group/category system allows us to define a set of user properties (the cat gories) and to assign a list of values (the groups) for each property. Independently of users, each category can be thought of as a reference list. Each property is related to a different reference list: a user's specialties are taken from the specialty category, a user's profession is taken from the profession category.

However, we may want to make more subtle distinctions and define many properties that use the same reference list. We could, for instacne, keep track of the user's medical specialty (what the user, as a doctor, is qualified in) and the user's interests (what areas of medicine the user likes to read about; we'll highlight articles based on this property). The reference lists for these two properties would be similar enough that we may want to use the same list.

Custom attributes

Since we can't anticipate all the user data we may want to store, we'll need to define custom fields as we go. The KOL custom info model should be sufficient for this purpose. The trick is to store the custom field definitions in the common database, so that if we add a field for a new site, we can apply it to the older ones as well. Fields that are truly arbitrary and site- or client-specific could have names prefixed with the site or client name.

With the use of SQL creation query, we could define groups based on custom attributes, if need be.

Access contr l

Access control is clearly site-dependent. A user can be an editor on theheart but have no rights whatsoever to influence content on jointandbone. Roles should then be associated to site profiles.
We can get a fair amount of flexibility by using roles and permissions, with each role implying a set of permissions. Permissions themselves can be simple strings (with their meaning embedded into the code), or structures made up of: an object type (news, poll, user), an action type (create, edit, delete) and possibly a scope or degree attribute (globally/locally, permanently/temporarily).

Preferences

We can express preferences as rules. They define how a site should behave. Some of those rules apply to the site as a whole (send monthly newsletters to the user); some apply to specific applications (display news highlights in inverse chronological order).
The set of rules that can be modified springs directly from the applications th mselves. We can't set rules such as display order for an application that has a hardcoded display order.
It should then be up to each application programmer to identify features that lend themselves to configuration, and to produce corresponding rule definitions. Whenever possible, similar rules in different applications should use similar data structures. Rule definitions, of course, may evolve from one version to another. This must be documented.
For maximum flexibility, we can use XML to set those rules, and set them at three different levels. Each application can have its own default rule definitions. Each site can then have its own default configuration, overriding some of the d fault application rules if desired. Each user can then have his own set of preferences overriding the site defaults. Additionally, a separate definition file for th site could list all applications and specify which may be configured and which may not.

Site communications

W do a fair amount of mass mailing on our sites, and will probably do more in the future. We need to keep track of what types of mass mailing may be p rformed for each site, and which communications each user agreed to receive. This could be perceived as just another preference, but since this data will mostly be used by an external system (the mass mailing software) rather than our site engines, we should make this a separate concept. This is best handled by creating a couple of database tables: a reference table listing all communications generated by each site, and an association table listing which communications ach user has agreed to receive. The mass mailing software can then consult these structures without having to resort to our site logic.

User relationships

There are a few reasons why we may want to define relationships between user profiles:

- to indicate that a user registered following the advice of another user (if we ever want to reward users who bring in more users)
- to indicate that a user has been invited by another user, and thus belongs to a work group centered around that user (see the "thought leader" concept developed for theheart.org)
- to let users build mailing lists for themselves (this should only be done as part of a messaging application, since there are a lot of users involved; do we really want to give our users the possibility to spam other users?)

For maximum flexiblity, we can create a qualified association table to link pairs of us rs. This makes it easy to define new relationships.

Admin

Load levels

When a user object is loaded in memory from the database, we don't necessarily need to load the entire profile in memory. Most user sessions are simple browsing sessions where the user's personal information is not needed.

We can define a minimal set of attributes needed for normal site operations. These would include the user's name, complete site profile, and most classification attributes. When a user logs in to a site, we need only do a partial load, storing only those essential attributes in memory. This keeps user objects smaller, allowing us to save some RAM on our servers. This may prove especially useful in high traffic periods.

For those few operations where a complete profile is needed, we can then do a full load. The user's complete profile will then reside in memory until session's end; we won't need to get that extra info from the DB more than once per session.

Version numbers

There's always a risk that two people could find themselves editing the same object at the same time. We can use version numbers to help us make sense of this.

Every record in the database would start with a version number of 0. Every time we want to save a modified object, we compare the version number in the database with the one in memory. If both match, we increment the number by 1 and save the object to the DB. If the numbers don't match, we can warn the user that the object has changed since it was opened.

This is the row version concept that first appeared in the VOICE model, if I remember well.

Stats

Unique identification

For various reasons, we cannot force a user to use a single person profile across *all* of our sites. We can, however, fake it for stats purposes.

In the database, each user profile will be attached to one person profile. Suppose a user creates three user profiles within one site family, each with its own separate person profile. It's a possibility, even though our interfaces will discourage this. We can, however, confirm that all three person profiles represent the same person. We can designate one of these as the primary profile, and add a "main person profile" attribute to each of the user profiles to point to that primary person profile. Most of our systems would treat the user's profiles as the user intended, but our stats reports could use the "main person profile" attribute to generate better results.

Common database

If we are to keep track of all our sites, group them into site families, and allow communications between them, we'll need a common database which will be used as a reference by all of our systems.
This database should contain:

- a list of our site families, with connection parameters for the user repositories of each
- a list of all our sites, each associated to its site family (and the organization(s) that own it)
- a set of policies defining which sites may import users from which other sites
- a list of organizations, including sponsors (and including Conceptis)
- a bunch of reference lists (medical specialties, geographical locations, interests, whatever we can think of, really)

The common database may also need to have its own list of persons so we can track contacts at various organizations. We can use this list as a global reference. This would be useful for mass mailing, since every site could define its own test mailing lists by picking Conceptis users from the global persons table.

Accessing the data

We can use Oracle techniques to export the reference lists to each site family's database in read-only mode.

MediaSurface

User mapping

How does all this affect our content management techniques? Basically, for each site, we will create a small number of MediaSurface user accounts, each representing a different level of access. Each site profile created for a user will be mapped onto one of these MediaSurface user accounts.
In other words, we will use MediaSurface for rudimentary access control. The content delivery engine of each site will handle another layer of access control, as well as classification and preferences.
Unfortunately, the MediaSurface engine doesn't allow us to explicitly associate user accounts to sites. We can't just query MediaSurface to get a list of users for a given site. For that reason, we should keep a list of all MediaSurface users and their site associations in our common database (see section 0).

Reference lists

For stats purposes, we must classify our users in a number of ways: profession, specialty, geographical location, etc. For personalization purposes, we must also apply the same classification to our content.
At this point in the development of the new framework, MediaSurface is the defining influence. We can't modify MediaSurface to make it interact with external systems, but we can code our systems so they will read the data defined in MediaSurface, and modify it if necessary.

MediaSurface alone

There are two ways in which we can define reference lists within MediaSurface.

Fields and value lists

Each classification category (profession, medical specialty, country, etc.) can be defined as a field for each item type where that category may be relevant. We must then define a list of valid values for each of those fields.
One problem with this approach is that reference lists are tied to item types. If we define a profession field for the "article" item type and for another type -- "event", let's say -- which is not an article subtype, we'll have to define a profession list for the article type and another profession list for the event type. Both lists will be independent: they will need to be updated separately, and will have no referential integrity between them. The only way we'll be able to match the "doctor" profession defined for articles and the "doctor" profession defined for events is by comparing spellings.
Another drawback is that MediaSurface may not do true item type inheritance. The software allows us to define an item type, say "Generic article", then define a child type (let's call it "MediaPulse") which will inherit all fields from its parent. However, it may be that MediaSurface simply creates the child type by copying

the parent, such that any updates to the parent may not be reflected in the child. If w modify the list of valid values for one the parent type's fi lds, the corresponding values in the child type may not change. [redacted]
The least we can do, if we use this technique, is to keep an external set of reference lists, if only in a static file somewhere. Whoever defines new item types would use those lists to populate new fields.

Item relationships

An alternative to the above technique is to give our reference lists a more concrete existence by defining an item type to represent each category (profession, specialty), and defining an item to represent each value (doctor, cardiology) within a category. We end up with a structure that mirrors the category/group system outlined above, although there can be no hierarchy of groups in this case.
When we want to associate a profession to an article, we create a relationship b tween the article item and the profession item. MediaSurface does not allow us to qualify relationships, but we can simulate that by storing the data in XML and interpreting it with our CMS driver. If need be, we could define multiple relationship types using the same reference list.
This technique may slow down content delivery since it adds another layer of code between MediaSurface and the delivery engine.
This is also problematic if we allow for multiple "fields" using the same reference list. In that case, we can't quickly generate lists of items with a certain value in a certain field, since for each item linked to the value, we must check whether the link pertains to the specified field or to another field.

Integration with user management

Given the scope of the work that must be done on the new framework, we can't expect to build a final, perfect system right away. Here are a few different approaches we could use. Note that we may be accessing the common database indirectly: subsets of the common reference lists may be "mirrored" at the site l vel to facilitate database operations.

MediaSurface-driven

- The Content department creates all reference lists directly in MediaSurface. The country list should be the same we've been using on recent sites (DUCCS KOL, for instance).
- Once the intial sets of values are in, we create them in the common database (using SQL creation scripts)
- From that point on, whenever somebody wants to add a value to a list, they must send a request to the DBAs, who will make sure it gets inserted correctly in both systems. To keep the lists safe, we may want to remove list edition privileges for most MediaSurface users.

Feeding MediaSurface from the common database

This depends on how we store the refer nce lists in MediaSurface. If we use field-based valu lists:

- We build a mapping tool to match MediaSurface item type fields to user management lists (and, possibly, to match individual field values to reference list values). We may allow for single-level mappings, that is: if we create a hierarchical specialty list in the common database, we may have a MediaSurface "specialty" field reference specialties of level 1, and a "sub-specialty" field referencing specialties of level 2. This can work as long as there's no requirement for an item's sub-specialties to match its specialties.
- We build a management tool for user management lists, and have that tool keep the MediaSurface lists up to date.

If, instead, we create each list value as a content item:

- We build a management tool for user management lists, possibly using the KOL group manager as a starting point. Reference values are treated as full-fledged entities, and the business logic that creates, modifies and deletes those entities updates the CMS simultaneously, using the CMS driver.

CAL-based integrity

- We create all reference lists in the common database. This will be our only point of reference
- We write the CAL so that all item edition forms take their reference values from the common database. When an item is saved, any relationships between the item and our reference values are simulated by storing the primary keys of the reference values in designated fields of the MediaSurface item.
- We do *all* our content management operations through the CAL.

Persons

Some person records will be stored as MediaSurface items so they can be associated to documents as authors, copyright holders, and so on. We may or may not want to store those person records in the common database as well. If w do, we'll have to keep them synchronized.

Use cases

Unless otherwise noted, any mention of the database (DB) makes reference to the database for the current site family.

Regular user experience

Registering on a site

Actors:

- a potential user

Pre-conditions:

- none

Special requirements:

- in order to offer an "import from another site family" option to the user (subflow 0), some information usage policies must be defined first.

Main flow:

- The user goes to the site entrance
- The system creates a session object
- The user clicks the "Register" link
- The system asks the user whether he already has a profile within that site family (or, just possibly, in another site family; we'll decide on a site-by-site basis whether or not to offer that option)
- If the user says yes, the system retrieves the user's profile (see subflow 0 for retrieval within the same family, or subflow 0 for retrieval from another site family)
- If the user says no, the system takes the user through a set of forms to create a new user profile (see subflow 0)
- The system checks to see if the resulting user object already has a profile for this site. If so, we can consider the user logged in. **end of flow**
- The system attaches the user object to the session object
- The system asks the user for site-specific information and creates a site profile in the database
- What follows depends on the site policies. Either:
  - the account must be validated manually before the user can do anything, in which case the system tells the user to wait for notification before trying to log in, or
  - the user goes through a probational period while we validate the account; the system flags the user's site profile accordingly, then loads that profile into the session object and lets the user navigate the site, or
  - the account does not require validation, or gets validated automatically and immediately; the system loads that profile into the session object and lets the user navigate the site

Post-conditions:

- the user now has a site profile for the current site

Subflow: Identifying an existing user profile (same family)

- The system prompts the user for a username and password
- The system checks the database for a corresponding user profile
  - if there is none, the system displays a "profile not found" message and gives the user a choice: try again (back to the top of this subflow) or create a new profile (switch to subflow 0).
  - if the profile is found, the system loads the user object in memory (including the site profile, if there is one)

Post-conditions:

- we now have a user object to work with

Subflow: Identifying an existing user profile (from another family)

- The system gives the user a list of sites from which he can retrieve his profile
- The user picks a site
- The system establishes a connection with the appropriate user repository
- The system prompts the user for a username and password
- The system queries the remote repository for a corresponding user profile
  - if there is none, the system displays a "profile not found" message and gives the user a choice: try again (back to the top of this subflow) or create a new profile (switch to subflow 0).
  - if the profile is found, the system:
  - loads the user object in memory (full load)
  - removes the ID numbers from the user object
  - edits the user object's history field to indicate where the user profile was imported from
  - saves the user object to the local database, generating new IDs in the process

Post-conditions:

- we now have a user object to work with

Subflow: Creating a new user profile

- The system prompts the user for basic personal information
- The user fills in the form
- The system checks to see if we already have one or more user records matching the given information
  - if so, the system lists the matching records by username and suggests that the user pick one
    - if the user picks one, the system prompts for a password.
      - if the user enters the right password, the system loads the user object in memory (including the site profile, if there is one). **end of flow**
      - if the user doesn't enter the right password, we go back a few steps to the username list
    - if the user chooses not to use an existing profile, we move on to the next form
- The system prompts the use for more complete information
- The user fills in the form

- The system places all th data in a person object, then asks the user to pick a username
- The user enters a username
- The system confirms that the username fits our rules, then checks for an existing account with that username.
    - if the username is already taken, the user must enter another username (we go back to that step)
- once the username is confirmed, the system gets a password from the user, creates a user object, and saves it to the database

Post-conditions:

- we now have a user object to work with

Logging in to a site

Actors:

- some user type of person, with no great distinguishing characteristics and having accomplished nothing of great import to this use case

Pre-conditions:

- none

Special requirements:

- none

Main flow:

- the user gets to the site
- the system creates a session object and saves it in the DB
- the user gets to the login page somehow (this varies from site to site; some sites have a public section that doesn't require any login)
- the user enters a username and password
- the system looks for an account by that name, then checks the password
    - if there is no match, the system displays a username/password error (we keep these vague, to avoid giving would-be hackers any clues as to what they're doing wrong). If the user has done too many of these unsuccessful login attempts, we switch to the "account lock-out" alternative flow (0). Otherwise, we start again from the top.
    - if the username and password both match, the system loads the user object (partial load) in memory, attaches it to the session object, and saves the session object to the DB again
- once the user's identity is confirmed, the system checks the age of the user's password against the site's password rules. If the password is expired, or about to expire, the system suggests a password change (see alternative flow 0).
- once the password issue has been resolved, the system displays the site's welcome page

Post-condition:

- the user is now logged in

Alternative flow: account lock-out:

- the user's account is locked (this may be done differently on different systems)

- the user is advised of the lock-out. The lock-out message should include tech support info.
- depending on the system, we may want to send an email warning to a Conceptis tech support address, or write a note to a log file. This allows us to monitor who's locked out and to provide assistance without any prompting from the user.
- the user cannot enter the site. **end of flow**

Alternative flow: password expiration

- if the password is set to expire within less than X days (warning period depends on the site), the system displays a warning and offers the user a chance to change the password now.
    - if the user declines the offer, we go back to the main flow
- if the user decides to change his or her password, or if the user's password is expired, the system displays a "change password" interface. If the password is not expired, a Cancel button will be shown, allowing the user to return to the main flow if desired
- the user enters the current password, then enters the new password twice
- the system checks the passwords for any irregularities.
    - if there's anything wrong with the passwords, the system displays an error message and displays the "change password" interface again. We go back a few steps
- if the passwords are correct and follow the site rules, the system reloads the user object from the database (just to be sure the information is current), changes the user's password, and saves the data to the database again.
- we go back to the main flow

Visiting a site (within the same site family)

Actors:

- a user

Pre-conditions:

- the user has an account on site 1, but not site 2
- the user is logged in to site 1

Special requirements:

- the policies defined for site 2 allow visits from site 1

Main flow:

site 1

- the user clicks a link pointing to site 2
- the system directs the user to site 2 and passes along the username, password, session ID and the ID code for site 1

site 2

- the system receives the login request from site 1
- the system searches the DB for a session matching the given session ID
- the system loads the session in memory into a session object and user object

- the system confirms that the username and password of the user object match the data sent by site 1

Post-conditions:

- the user is now logged in to site 2

Visiting a site (cross-family)

Actors:

- a user from one of our site families

Pre-conditions:

- the user has an account in site family A, but not in family B
- the user is logged in to site 1 in family A

Special requirements:

- the policies defined for families A and B must allow for visits

Main flow:

site 1

- the user clicks a link pointing to site 2
- the system directs the user to site 2 and passes along the username, password and the ID code for site 1

site 2

- the system receives the login request from site 1 and creates a session object
- the system checks the site policies to confirm that the visit is allowed
- the system calls the user management module for site family A and passes the username and password as parameters
- the system receives a user object
- the system creates a site profile in the DB, loads it into the user object, and attaches the user object to the session object

Post-conditions:

- the user is now logged in to site 2

Profile modifications

Actors:

- a user of any race, gender or creed

Pre-conditions:

- the user is logged in to one of our sites

Special requirements:

- ?

Main flow:

...

Accessing a personalized application

Actors:

- a user who expects preferential treatment

Pre-conditions:

- the user is logged in to the site
- the user has defined preferences in relation to the application

Special requirements:

?

Main flow:

- the user requests the application
- if a set of preferences has already been cached for this application and this user, we skip ahead to the last step
- the system checks the load level of the user object
- if the object is not fully loaded, the system loads the full user data
- the system retrieves the user's preferences regarding the application
- if the user has not set every possible preference for the application, the system loads the remaining preferences from the site defaults defined for the application
- if the loaded preferences don't cover the full set of possible preferences, the system completes the set by reading the application's defaults
- the system caches the completed set of preferences
- the system calls up the application and passes it the preferences

Post-conditions:

- the complete preferences for the application, for the given user, are cached in memory
- the application behaves according to the user's preferences

User management

Searching the user database

Deleting a user

Merging user accounts

Basic profile

Here's an attempt at defining a user profile that will fulfill our needs. Attributes preceded by an asterisk are uncertain: they would be useful, but may not be worth incorporating in our data model.
Some ideas to consider:
multiple addresses?

Person attributes

- **name**
  - title/prefix/salutation (e.g: "M.", "Dr.")
  - first name
  - middle initial(s)
  - last name
  - suffix (e.g.: "III", "Jr.")
- **demographics**
  - gender
  - date of birth
  - degrees/education level
- **location/contact info**
  - location name
  - location type (hospital, university, corporation, etc.)
  - street address
  - city
  - state/province
  - country
  - zip/postal code
  - phone number and extension
  - fax number
  - pager number
  - email addresses
- **classification**
  - affiliations (qualified by role/title in each organisation)
- **meta**
  - preferred email type (HTML or plain text)
  - notes
  - profile creation date
  - created by
  - profile modification date
  - modified by

Some notes:
affiliations: this would imply the creation of an organisation directory, which we could use to track sponsorships as well
notes: this field is to be used by user managers as a scratchpad. If a user often gets locked out, we can write it here. If we had to change the user's profile recently, we can write it here.
email addresses: apparently, many people create multiple accounts on our sites so they can receive the site newsletter at multiple email addresses. Th new user

profiles should support multiple addresses for each person, and possibly d fin simple rules for the use of those addr sses.

User attributes

General user attributes

- username
- password
- password hint, or "lost my password" questions and answers
- preferred email type (HTML or plain text)
- *preferred contact method (email, phone, snail mail)
- foreign ID (identifier from client)
- **meta**
    - account status (valid, on hold, etc)
    - notes
    - history
    - number of unsuccessful login attempts so far
    - profile creation date
    - created by
    - profile modification date
    - modified by

Some notes:
history: this may best be handled as an XML structure. Here we can keep track of a user's origin: if they were transferred from another system, we can store the name of that system, along with their user ID on that system. We can also store the circumstances in which the user registered: at the ACC conference, or the AHA conference, or any other one.
username: we should agree on a standard username syntax to be used throughout our systems. This can save us a few headaches, especially in cases where we need to duplicate the accounts in secondary systems (the Merck CTC, for instance, uses Windows NT accounts and creates accounts on an Authentica server). My suggestion: length between 4 and 20 characters, using only lowercase letters and digits
password: as with usernames, we need a global standard. We may need to adapt that standard for sites using parallel accounts in different systems. A good basic set of rules would be: minimum of 6 or 8 characters, allowing for letters, digits, some basic punctuation, but no spaces. Passwords would be case-sensitive for better security.

Site-specific user attributes

- accepted mail message types
- user type
- roles
- foreign ID (identifier from client)
- preferred language(s)
- preferences
- **m ta**

- notes
- profil creation date
- created by
- profile modification date
- modified by

Some notes:
accepted mail message types: this is where we track what kind of communications the user agrees to receive from the site: newsletters, special offers, and so on.
user type: this is attribute is meant for stats purposes, so we can exclude certain types of users from our stats
preferences: could be stored as an XML block using a simple module/rule structure:

```
<prefs>
<news>
<ordering>date</ordering>
<number_shown>20</number_shown>
</news>
<right_column>
<calendar>show</calendar>
<highlights>show</highlights>
<feature>hide</feature>
</right_column>
</prefs>
```

Privacy issues

<u>Protection of personal information</u>

The issue

Many users like to limit the amount of personal information they provide to one website or another. They may want to prevent companies from prying into their private life and habits, or restrict the volume of email solicitations they receive.

Personal information can be used at three levels:

- it can be used within the site, to influence content delivery and generate traffic stats
- it can be passed on to the client for whom the site was built, when applicable
- it can be passed on to third parties for various purposes: accreditation, census, registration to other sites, and plain old spam

We must address all three levels.

Solutions

Privacy policies

The least we can do is have clear, explicit privacy policies on each of our sites. In many cases, we will have to coordinate with a client in this regard. Some clients have their own privacy statements already drafted: in those cases, we should make sure that we can indeed respect the client's policies.

Password security

Most (or maybe all) of our systems store passwords without encrypting them, so that anybody with read access to the database can retrieve all of our user's passwords. This is a security risk: if somebody were to hack one of our databases, all accounts would be compromised.

We can solve this by encrypting passwords prior to storing them, using a one-way encryption technique. During the login process, we take the password supplied by the user, encrypt it, and compare it to the password stored in our database. The main drawback of this approach is that we couldn't offer password retrieval features for users who forget their password (unless we tied the encryption key with the retrieval process somehow).

Another solution would be to hide the password field in the database, making it readable only by a few people. Our applications could still read it, but users accessing the database directly would not see the passwords.

Some sites have special requirements in this regard. The Merck CTC, for instance, is a mirrored site which uses Windows authentication. Every time we create a new user profile, a corresponding Windows NT user account is created on the server and on th mirror as well. If one of the machines crashes badly enough, we may need to rebuild that machine's collection of Windows user accounts. In order to do that, we need access to the passwords.

Usage stats

The issue

In order to get a better picture of how our sites are faring, and to fine-tune our applications, we may want to generate cross-site usage statistics. How many jointandbone users are also registered on theheart? How and why do users switch from one site to the other? We can find many advantages in knowing that user X on site 1 and user Y on site 2 are in fact the same person.

On the other hand, the users themselves may not want to be followed around and scrutinized in such a way. They may want to have distinct, independent identities on the various sites they visit. Either they prefer to receive their various newsletters at separate addresses, or they're worried about us accumulating too much information about them, or they find it more secure to keep separate sets of usernames and passwords.

Solutions

Voluntary and involuntary identification

Our sites, by their very nature, give us one advantage. Most sites we build are targeted at medical professionals and promote an exchange of ideas between said professionals. For that reason, our users have little use for anonymity or cool nicknames. We can probably convince many of them of the benefits of having a single account usable across most of our sites.

While migrating our existing sites to the new framework, we should give users the opportunity to combine their accounts. All we need to do is come up with a standard interface for doing so, and write simple account converter for each site.

We may also attempt to centralize user information ourselves by comparing accounts on various sites. We could consider two accounts to be representations of the same person if we get an exact match for the first name, last name and email address, for instance. For each group of matching accounts, we could create a single person profile. We should, however, differentiate these from "official" person records and differentiate between the two when generating stats.

Site families

We can make the "unique profile" concept clearer and more reassuring by grouping our sites into families. We could, for instance, group into one family most of our specialized medical portals such as jointandbone and theheart. We could group other sites by client: STICHtrial and DUCCS, since they both belong to Duke, could share the same users. By default, a user would have one profile that would be used by all the sites in one family.

We may need to build those families not only around themes, but also around security concerns. If a user were to use the same profile for a highly secure site (such as a CTC) and for a low-security site, it may become possible for a hacker to crack the user's account on the low-security site and use that information to get on the high-security site.

Internati nalization

Acc unt matching

As previously mentioned (see sections 0and 0), we need to encourage users to set up a unique profile on our systems. We can automate this to some extent:
We shouldn't, however, merge similar accounts automatically, without user intervention. There are a few reasons for this:

- it's hard to be absolutely sure of a match. We can't merge by email address alone, since it's relatively common for two people to share a single address. The combination of first name, last name and email address is a better criteria, but there's at least one scenario where two accounts that seem to match could still represent two distinct people

- if we mistakenly merge the accounts of two different people, we get a security breach if one of these two people had fewer permissions than the other. This endangers the site data, and there's also a risk of sending confidential personal information the wrong person

- even if we do get an automatic match system going, it will only be handle a small percentage of matches; most of them will still have to be confirmed manually

Basic matching technique

We need to write two pieces of codes: one to find matching accounts, and one that can merge a small group of accounts together.
The account matcher should search through a set of user profiles (all users of a site, or all users within the entire system) and return a grouped list of potential duplicates. We can run the account matcher once when we first transfer all the users over to the new framework, then set it up as a scheduled task.
The account merger would combine all data from a group of duplicate accounts and store that data into one of those accounts (designated as the primary account). The other accounts would be deactivated.

Matching within sites

It's especially important that we match accounts within each site, at least. There's no reason why a user would need to create and maintain multiple accounts within a site.
We can enforce this in a few ways:

- send automated emails to owners of duplicate accounts
- provide a form to allow owners of duplicate accounts to merge their accounts without our help
- flag duplicate accounts so that users g t a reminder when logging in

Matching acr s sites

Using unique accounts across sites is more touchy. It should probably be restricted to families of sites, for security reasons.

[redacted]

Stats

<u>The distinction between stats and operations</u>

<u>The data warehouse</u>

Uses for the person object

<u>Contact info</u>

<u>Authorship</u>

Modules and interfaces

Site-based operations

Security module

Standardized login form

Change password form

Registration/profile edition interface

Registration forms will generally have to be custom-built, as there tends to be too many subtle differences between the registration processes of different sites. We can simplify building those forms by creating custom tags that will generate form fields for certain parts of the standard profile (address, email preferences, etc.)

User management module

Search interface

Editing interface

Account merging interface

Client interface

A simplified version of the editing interface can be made available to the client. This is a Good Thing because:

- the client can create and maintain user accounts at any time, without us having to do any work
- user accounts entered in this way always conform to our standards (this is the main advantage, really)
- it doesn't leave any ambiguity as to where the master database is (compare with the Merck CTC, where the client is building an Access database and will be sending us the data, while we will handle users' requests for profile changes).

Stats module

Common database

Reference list management interface

Site management interface

Since we seldom create new sites or site families, this interface wouldn't see much use. Additionally, such an interface would make it easy to screw up the operation of the entire framework. For those reasons, it may be best to maintain the sites table entirely through database change requests.

Questions

Q- Could we force people to have a unique profile (person and user) across all Conceptis sites?

Q- How de we handle personal information when people from one site family visit sites in another family?

Q- What personal data do we especially need to collect for each site?

Q- How do we handle the distinction (or lack thereof) between site users and systems users?
A- MediaSurface users and site users will be different. Each site user will be mapped onto a MediaSurface user representing a role (more or less).

Q- How do we reconcile MediaSurface reference lists and user management reference lists?

Q- Can we store locations as groups even though locations require a special treatment?

ANNEX 2

THE CONCEPTIS DELIVERY DESIGN

by Sam Guembour and Thomas Bennett

This document provides a comprehensive architectural overview of Conceptis' Content Delivery Architecture. It is intended to capture and convey the significant architectural decisions that have been made during the process as well as to provide the framework through which the delivery engine will be implemented.

22 May 2002, Version 1.0

TABLE OF CONTENT

1. INTRODUCTION 69

1.1 Revision history 69

1.2 Terminology 69

1.3 References 70

1.4 Requirements 70

1.5 Contacts 70

2. CONCEPTUAL DESIGN 71

2.1 Introduction 71
2.1.1 *Overview*.....71
2.1.2 *Purpose*.....71
2.1.3 *Scope*.....71

2.2 Architectural Goals and Constraints 72
2.2.1 *Considerations*.....72

2.3 Architecture 73
2.3.1 *MVC-2 Model*.....73
2.3.2 *Struts Implementation*.....76
2.3.3 *Physical Component View*.....78

2.4 Website Implementation View 79

2.5 Use-Case View 80
2.5.1 *Use-Case Realizations*.....80

3. PHYSICAL DESIGN 84

3.1 SCOPE 84
3.1.1 *Uses Cases*.....84
3.1.2 *Problem*.....85
3.1.3 *Solution*.....85
3.1.4 *Teams*.....85

3.2 THE STRUTS FRAMEWORK 85
3.2.1 *Struts Components*.....86
3.2.2 *Sequence diagram*.....87

3.3 WEB SITE DESIGN 88
3.3.1 *The scenario*.....88
3.3.2 *Business logic*.....94

3.3.3 *Security*....96
3.3.4 *Stats*....96
3.3.5 *Presentation*....97

3.4 **STANDARDS** 103
3.4.1 *Struts configuration*....103
3.4.2 *Packages*....104
3.4.3 *URI*....106
3.4.4 *Folders*....106
3.4.5 *Internationalization*....107
3.4.6 *Deployment*....107

4. CONCLUSION 108

TABLE OF FIGURES

FIGURE 1: DOCUMENT SCOPE 71
FIGURE 2: STRUTS IMPLEMENTATION 77
FIGURE 3: REQUEST USE CASE 84
FIGURE 4: ACTION CLASSES 90
FIGURE 5: FORWARD FLOWCHART 94
FIGURE 6: CMSDRIVER FOR MEDIASURFACE 95
FIGURE 7: SAMPLE CONTENT OBJECT 95
FIGURE 8: JSP TEMPLATES 97
FIGURE 9: TOP LEVEL PACKAGES 104
FIGURE 10: COMPONENTS VIEW 105

INTRODUCTION

When we start to think about a web site, we always need to deliver some content to a user and this one could choose different approach to read it.
With Mediasurface we know that the content is manage in side but there is no side to present it.
With this document, you will learn how to develop a site with a minimal development and with a very robust mechanism to handle it.

Revision history

Revision	Author	Description of Changes
Initial	Sam Guembour	
May 14, 2002	Sam Guembour	Addition for Struts configuration usage
May 14, 2002	Sam Guembour	Merge with Thom's document
May 22, 2002	Sam Guembour	Add section 3.3.3 Security
May 24, 2002	Sam Guembour	Add section 3.3.4 Stats

Terminology

MVC	Model-View-Control
JSP	Java Server Page
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

R ferences

[4] Struts, an open-source MVC implementation (http://www-106.ibm.com/developerworks/ibm/library/j-struts/)

[5] Struts User's Guide (http://jakarta.apache.org/struts/userGuide/)

[6] More About Struts (http://www.husted.com/struts/)

[7] Struts Console (http://www.jamesholmes.com/struts/console/)

[8] Struts Layouts (http://struts.application-servers.com/)

[9] Using Taglibs in Dreamweaver UltraDev (http://jakarta.apache.org/taglibs/doc/ultradev4-doc/intro.html)

[10] Conceptis Development Standards by Joan Roch

Requirements

[2] Struts Framework Library (struts.jar)

[3] Resin Java Web container (http://www.caucho.com/products/resin/)

[4] Conceptis API for Mediasurface and services

[5]

Contacts

Thom Bennett	Conceptis Chef Analyst
Eric Hechinger	Project Manager for Mediasurface project
Sam Guembour	Chef Architect and Development Department Director

CONCEPTUAL DESIGN

Introduction

Overview

In Fall 2001 Conceptis purchased the Mediasurface content management software to manage all aspects of the content publishing flow. Although the Mediasurface software provided the facility to manage the delivery of content over the web, it was determined that a more robust solution would be required to handle Conceptis' business needs. These needs include: managing a shared portal membership registry; provide reporting in the form of usage statistics; provide an infrastructure whereby the look and feel of a website or portal could be maintained separate from the programming of the website and at the same time share the programming, or business logic, between the portals.

Purpose

This document provides a comprehensive architectural overview of Conceptis' Content Delivery Architecture. It is intended to capture and convey the significant architectural decisions that have been made during the process as well as to provide the framework through which the delivery engine will be implemented.

Scope

This document will be used in subsequent development by defining the high-level packages and interactions between components and external sub-systems, in particular, it will focus on the Content Delivery and the Shared Business Logic within the overall Conceptis Technological infrastructure, depicted below in Figure 1.

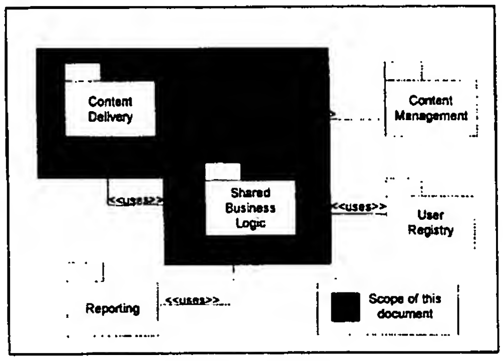


Figure 1: Document scope

Archite tural Goal and Constraints

Considerations

The Conceptis Delivery architecture must provide a delivery "framework/architecture" that allows us to :
Describe/define a "content view" (as a website) without the need of advanced programming, e.g. by using design templates.

- Better manage developers/resources (skills, availability, ...)
- Designers
- Programmers

Address the evolving nature of the display of content according to a customers changing needs in a timely manner by providing a mechanism to better define a customers requirements ;
Associate the productivity with efficiency in the creation of websites.
The above considerations can be best addressed using an MVC-2 architecture approach, which decouples the **presentation** aspect of website content delivery from the **business rules** used to access the content, for example from a content management system, and the **control mechanisms** by which the business rules are invoked.
With an MVC-2 architecture the Conceptis delivery framework will:

- Use a dispatcher to handle preliminary context setup for incoming website requests, such as the session and user information;
- Use of a "Request Container" concept to manage the coordination of website presentation and business logic. The request container existing either as a servlet, passing control to a JSP template, or can be implemented directly as a JSP;
- Allow for the presentation portion being controlled by JSP templates referencing custom tags that implement the decoration components, buttons, backgrounds, etc...;
- the business logic being implemented by a combination of custom tags and Java beans or class files.

The above approach ensures the overall website presentation logic is relegated to the R quest container. Different views of the same websit can be obtained by

defining various "Request Containers" and associated "Layouts" while keeping th same cont nt.
Each piece of content for the site will itself require a separate business logic and corresponding presentation logic. We introduce the concept of a vBean and a contentBean to reflect this: the vBean being the presentation portion and the contentBean being the business logic portion.
The de facto standard for such an architecture can be found in the Struts framework (http://jakarta.apache.org/struts/index.html).

Architecture

MVC-2 Model

The overall delivery flow implemented using an MVC-2 approach is described below here referring to the figure below (see below 3.3 for actual Struts framework dispatching explanation) :

13. The **Request Dispatcher** receives all HTTP

14. It is the Request Dispatchers responsibility to ensure a **sessionContext** object exists for the particular website request, and if not take appropriate action, for example pass control to the "Login" module.

15. The sessionContext consists of all previously defined beans in their previous state ensuring the sessions persistence between calls as well as the member's information as obtained via the "**User Registry**" service. The sessionContext also includes a reference to the **ContentAccessService** which will proxy all request for specific content by subsequent business logic.

16. The Request Dispatcher invokes a url-mapping logic to determine an appropriate RequestContainer that will coordinate the actual response back to the users browser. The Request Dispatcher will also determine whether the incoming request requires a container or not, in the case of requests for "Decorations".

17. The **Request Container** receives control at this point and is responsible for coordinating the invocation of any services or beans. It does this by interpreting 2 JSP files, the first JSP file functions as a layout manager defining place holders for various component functions, or **vBeans**, such as the navigation menu, banner, poll, etc... The second JSP file includes the actual references to the JSP objects themselves (*The Request Container is an*

*implementation of a **JSP Template***)

18. The Request Container passes control to each of the referenced vBeans.

19. Each vBean receives control and processes the request accordingly. Each vBean implements 2 main functions: presentation and content. The presentation portion of a vBean will usually be implemented in JSP (possibly as a JSP template) and will reference the associated **contentBean** to get access to the content to be displayed.

20. The contentBean will reference the **contentItem** in the **ContentAccessService** package.

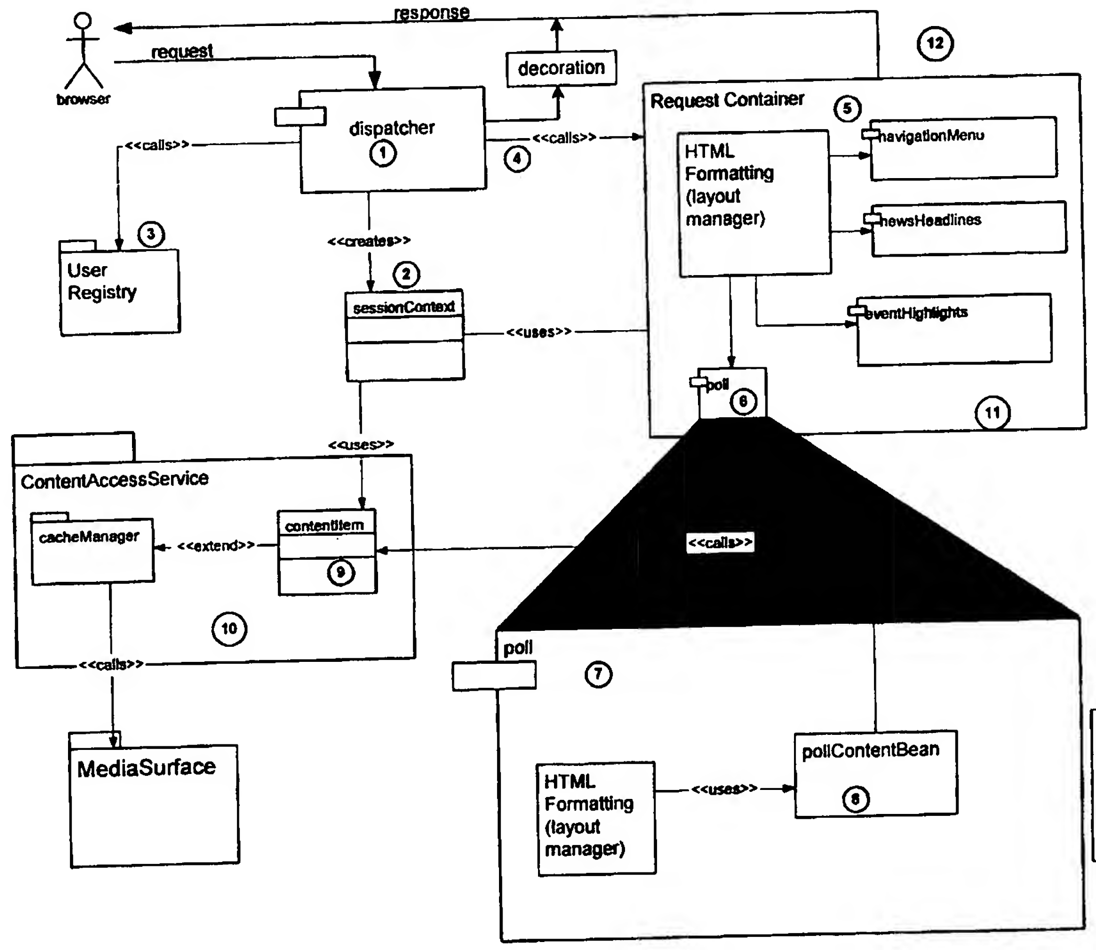
21. The contentItem will, itself, determine whether to invoke further logic or to access the content directly if it determines the particular content item already exists.

22. It is the responsibility of the ContentAccessService to provide any caching requirements between the business logic layer and the Content Management System interface (Mediasurface , for example).

23. The Request Container process all referenced vBeans similarly (6-10)

24. The Request Container returns the response to the requesting browser .

Sample request life cycle



The RequestContainer manages the presentation by providing holders for the various vBeans, it does this by using the JSP template concept. One JSP file determines which beans will be called or instantiated and another JSP file defines the template, in the form of HTML which includes calls to the template library. In this case a poll has been included on the page, the Request container passes control to the poll vBean which instantiates the pollContentBean gaining access to the content via the contentAccess object which manages the content request (as well as any caching logic). The users session object is in scope and responds to any personalization demands by the active objects.

Business logic is handled by the content beans, pollContent, in this example.

Struts Implementation

In a Struts implementation the dispatching is handled by th ActionServlet class which can b used, for the most part, as is. The ActionServlet implements a "forward" feature that is configured in the "struts-config.xml" file.
R fer to Figure 2 below for the following description of a sample request sequence:

11. The request is delivered to the Action Servlet which has read the "struts-config.xml" that defines the URL mappings and their forward actions.

12. The Action Servlet simply invokes the appropriate Action class for the given request.

13. Each Action class extends org.apache.struts.Action and also the com.conceptis.user.session class

14. The invoked action class ensures that a session object exits or can be created from the request url and parameters, invokes any generic business logic, stores session beans, and returns control to the Action Servlet.

15. The Action servlet redirects the result from the invokes Action class according to the return result, or "forward", possibly sending the user back , for example, to login.jsp.

16. In the case of a return of "success" the Action servlet passes control to a "Container" JSP which defines what other components need to be invoked or executed.

17. The Container JSP is essentially a JSP template that chooses the appropriate beans to be placed in the Layout Manager, i.e. that the results of home_page.jsp are to be output into the MAIN part of the Layout, equally it may decide that display_)news_article.JSP will be displayed in MAIN.

18. The Layout Manager is now responsible for assembling the output from the various component invocations by invoking each <template:get ....> according to what the Container JSP has previously decided.

19. Each component invocation essentially is a call to a vBean which may use previously executed business logic. The vBean is now responsible for outputting the result of subsequent contentbeans. For example in the case of News, the vBean will make a call to generic NewsContent Business Logic (BL), which may use any session objects, e.g. the User object in the case p rsonalization needs. The vBean is r sponsible for assembling the output from the busin ss Logic...

20. The process continues for each component of the Container ....

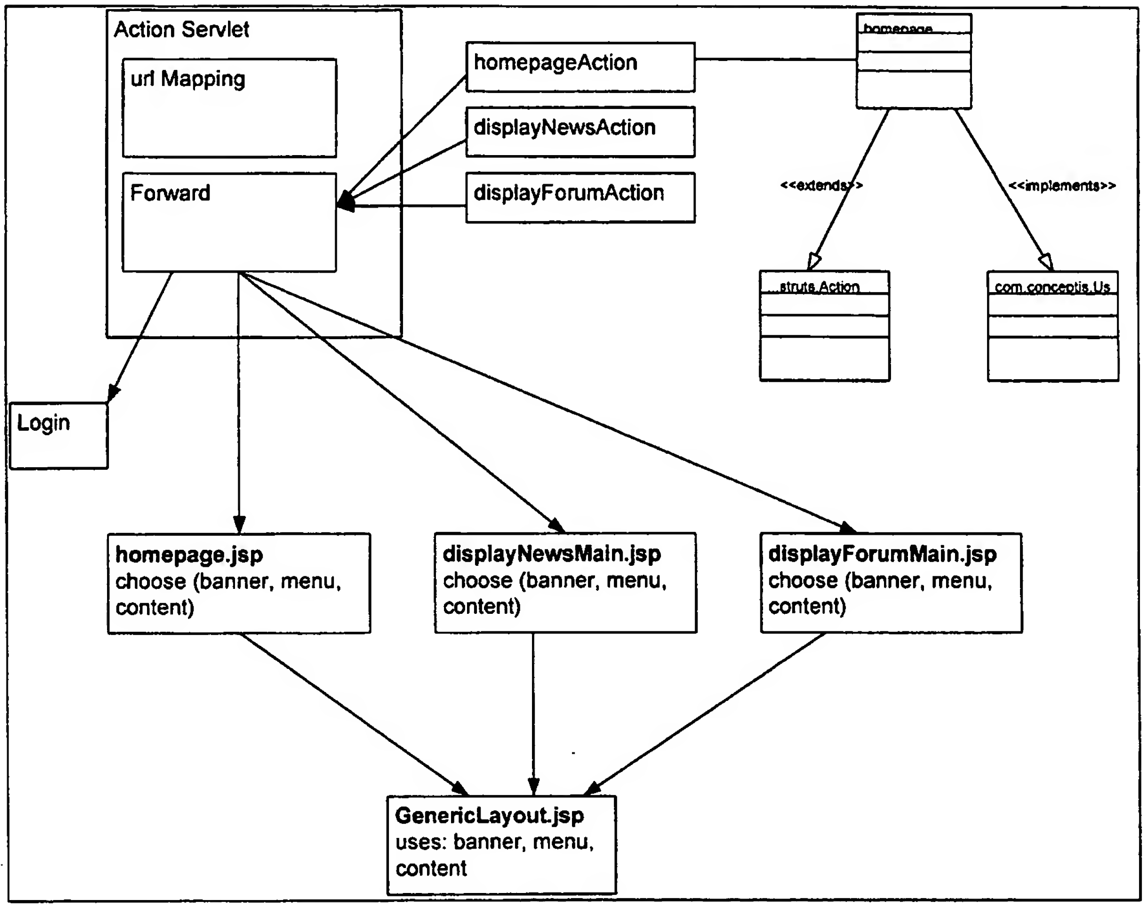
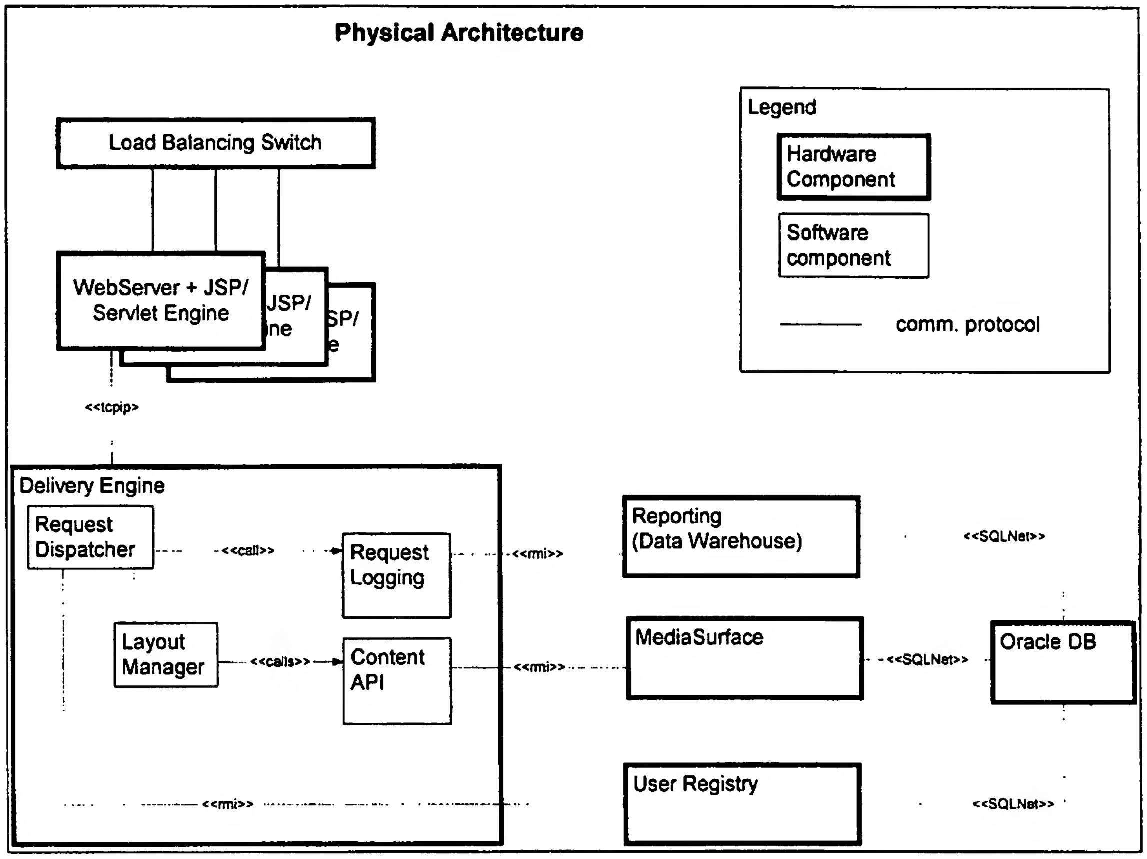


Figure 2: Struts Implementation

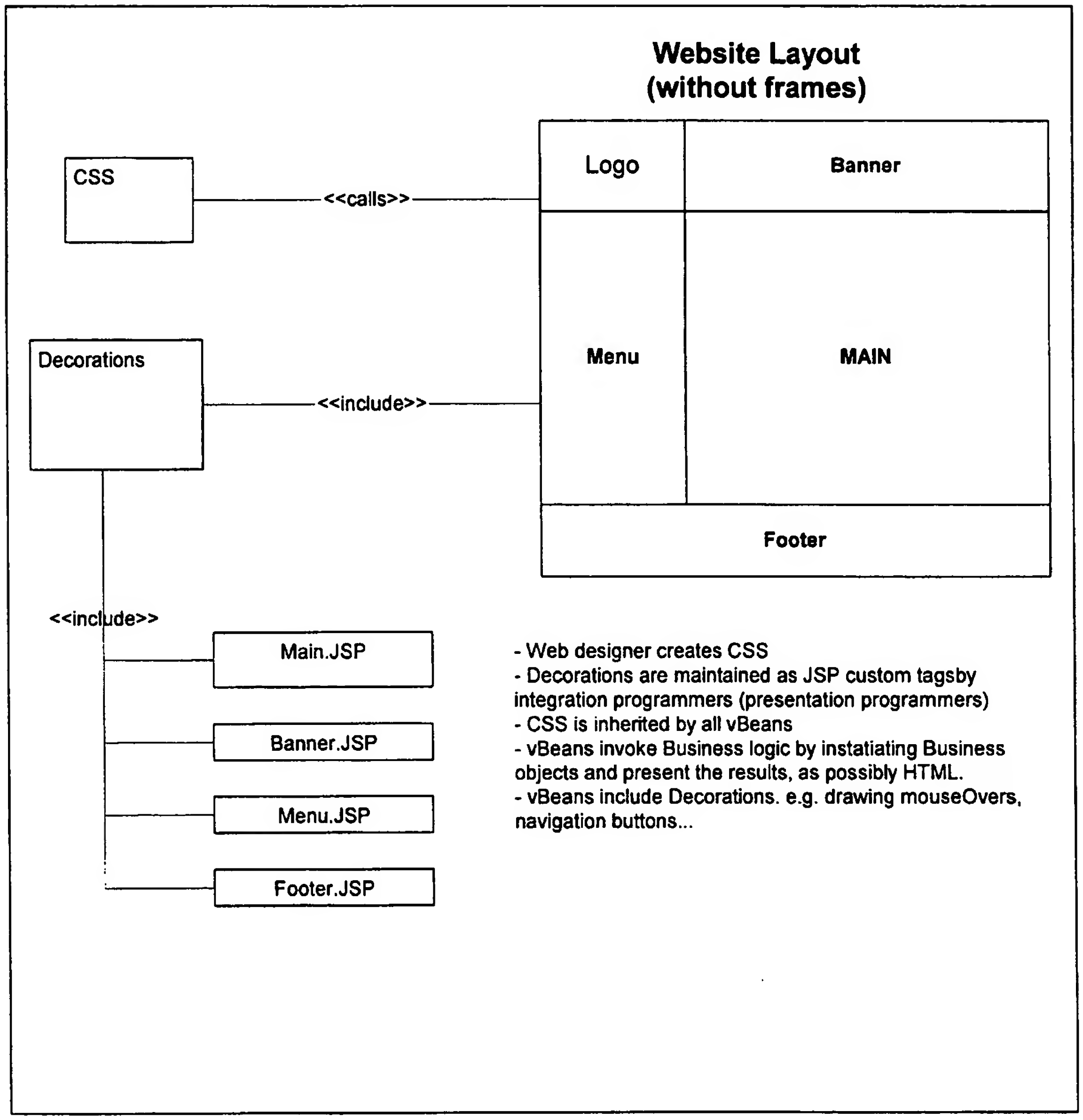
Physical C mpon nt View

System redundancy and scalability can be provided by distributing the services over multiple servers as outlined below. In the case of using a hardware based load balancing solution, it must support a cookie based session persistency (JSESSIONID as defined in the Java Servlet 2.2 specification).



W b ite Impl m ntati n View

The following depicts the relationship between the various JSP layouts and decoration objects. A cascading style sheet is used and will be inherited throughout and JSP custom tags will be used and included in subsequent JSP components to reference buttons, backgrounds, borders etc ...



- Web designer creates CSS
- Decorations are maintained as JSP custom tagsby integration programmers (presentation programmers)
- CSS is inherited by all vBeans
- vBeans invoke Business logic by instatiating Business objects and present the results, as possibly HTML.
- vBeans include Decorations. e.g. drawing mouseOvers, navigation buttons...

U e-Case View

Use-Case R alizations

Process Generic Request – Use case

Description

All incoming HTTP requests to the Conceptis website delivery engine will utilize this use case. In very general terms this use case represents the main controller component in the traditional MVC-2 architecture implemented using the Struts framework.

Actors

Client Browser
User Registry
Access Logging service

Preconditions

The Action Servlet has been initialised
The Action Servlet has loaded a table of "url-mappings" that will be used to handle the requests according to a given url pattern, e.g. it will be possible to create a mapping whereby all url requests ending in "*.do" are passed to a given Java Class.

Special Requirements

The users browser has enabled cookies

Main Flow

An HTTP request is received by the Action Servlet.
The Action Servlet extracts the URL from the request object and invokes the corresponding servlet class.
The servlet extracts the JSESSIONID variable from the request object
The servlet attempts to load the Session object, if the servlet cannot load the session object "Exception E1 – Invalid or missing Session object" is executed and return code of "failure" is sent back to the Action Servlet.
At this point the servlet has loaded a valid session object.
The servlet executes any generic business logic and returns a code of "success" to the Action Servlet.
The Action Servlet process the return code and invokes the appropriate JSP or Class file.
Control is passed to the appropriate JSP or Class file which will manage the remainder of the request and subsequent response. (see Generic Request Handler Use Case below)

Sub-flows

none

Alternative flows

none

Exceptions

Exception E1 – Invalid r missing Session object

The servlet was unable to load the session object associat d with the JSESSIONID either because this is the first request for a users s ssion or the user has turned off cookies during an existing session.

The servlet creates a new session object for the request and stores the requested URL in it.
The servlet sets the return code to "login"
The Action Servlet receives control and forwards the request to the Login service, invoking the "Logon – Use Case"

Post-conditions

A valid user session object is created within the applications scope

Generic Request Handler – Use case

Description

The Action Servlet redirects requests to a Request Handler according to a return code and a url-mapping define in the "struts-config.xml" file. It is the requests handler's responsibility to coordinate the output to the users browser. It is the request handler that controls the flow of output into the Response object. **The Request Handler will take th form of a JSP template** which forms the output container for the invocation of the various beans or applications for the given request. The request handler defines where each of the different aspects of a given page will be placed and so functions as a layout manager. A typical Request handler would take the form of:

Actors

Client Browser

Preconditions

A valid request object has been created
A valid user object has been created in the sessions context

Special Requirements

org.apache.struts

Main Flow

The request handler receives control from the Action Servlet.
The request handler immediately creates the response object
The request handler creates a decoration object according to the user object, for example selecting the appropriate style-sheet for the user and website that is being invoked
The request handler invokes each of the service functions which in turn invoke the appropriate business logic and presentation logic to include the results in response object.

Sub-flows

none

Alternative flows

none

Exceptions

Exception E1 –

none

Post-conditions

Log n– Use case

Description

The Logon use case is invoked either because the user has specifically referenced the Login Use case in the requesting URL or is being invoked as a result of a forward action from the "Process Request" use case.

Actors

Client Browser

Preconditions

In the event that the Logon use case is being invoked by the Process Request use case a Session object will already exist containing the originally request URL

Special Requirements

none

Main Flow

An HTTP request is received by the servlet.
The servlet extracts the URL from the request object
The servlet extracts the JSESSIONID variable from the request object
The servlet attempts to load the Session object, if the servlet cannot load the session object "Exception E1 – Invalid or missing Session object" is executed
The servlet

Sub-flows

S1 – Member List

none

Alternative flows

none

Exceptions

Exception E1 – Invalid or missing Session object

none

Post-conditions

Create Session Object – Use case

Description

All incoming HTTP requests to the Conceptis website delivery engine will utilize this use case. In very general terms this use case represents the main controller component in the traditional MVC architecture, and, as such, will be most likely implemented as a servlet running within a servlet container.

Actors

Client Browser
Personalization Server

Preconditions

Special Requirements

Main Flow

An HTTP request is received by the servlet.
The servlet extracts the URL from the request object
The servlet extracts the JSESSIONID variable from the request object
The servlet attempts to load the Session object, if the servlet cannot load the session object "Exception E1 – Invalid or missing Session object" is executed
The servlet

Sub-flows

S1 – Member List

Alternative flows

none

Exceptions

Exception E1 – Invalid or missing Session object

none

Post-conditions

PHYSICAL DESIGN

SCOPE

Uses Cases

Request Handling

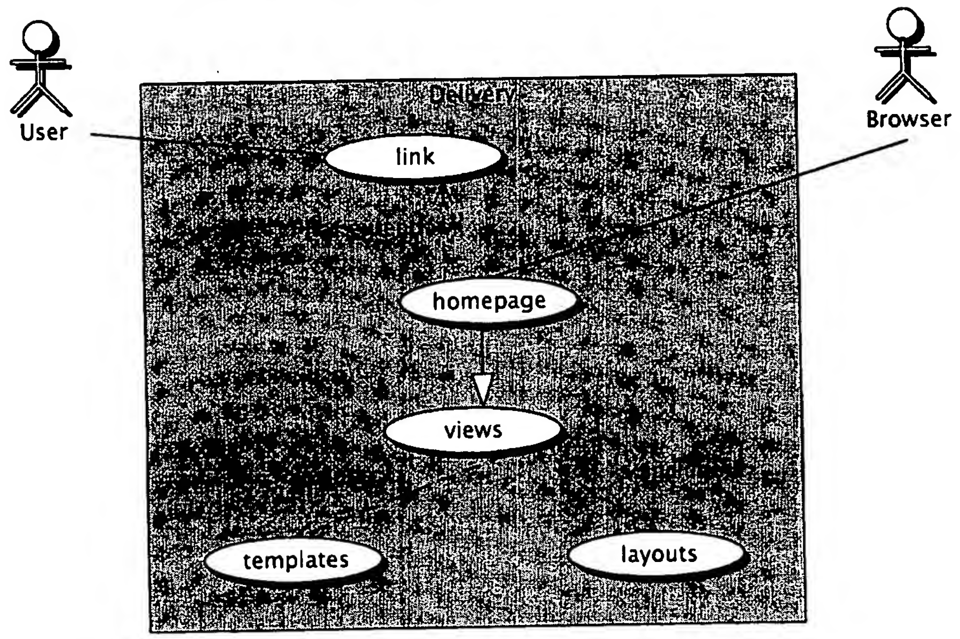


Figure 3: Request use case

- The user asks the homepage from a web link.
- The browser prepares the homepage from views.
- Views are created from templates and layouts.

Problem

When developing web application, we need to separate th application in 3 parts to build it.

1. Presentation
2. Logic
3. Persistence

Because different teams can develop these parts, we need to create an architecture that handles all parts of the application but can be developed separately and assembled in the deployment process.
With the Java programming language, we can make a web application that handle all this needs but there is no methodology describes in the language to separate these 3 parts.

Solution

By using the MVC pattern (model-view-control), we can handle all the programming needs but it's not enough to make a team works separately. It will be more useful to use a simple framework based on the MVC Model 2 pattern like Struts. http://jakarta.apache.org/struts/doc-1.0.2/index.html

Teams

To build a web application, we need to know how user will use this application. We will need to create some scenarios to describe the functionalities of the application and it's interaction with other systems. Also, creators of the application can choose different visual approach to present the data stored in the system.
Because all the jobs could be done separately in the development process, the framework that handles the entire component must permit to, for example, a programmer to implement all the actions that a web user can make on the site. This job is it self a part of the pattern and can be resolved by using the frameworks to extends all the actions and there configuration scenario.
Also, when a programmer needs to implement the content delivery, he must care about the logic to get it and to send it. By separating this business logic in it's own package, the programmer can focus on this logic without knowing where the logic is used. For this part, we can put all the business logic in application that handles all the web services.
At the end, a programmer can focus on the presentations of the content by creating the templates for all this king of content. The presentation is always a challenge for a web developer because there is a lot of possible change in the life of the application. Programmers must focus on separating all this views in small piece by using layouts.

THE STRUTS FRAMEWORK

Struts is an implementation of the MVC Model 2 pattern.

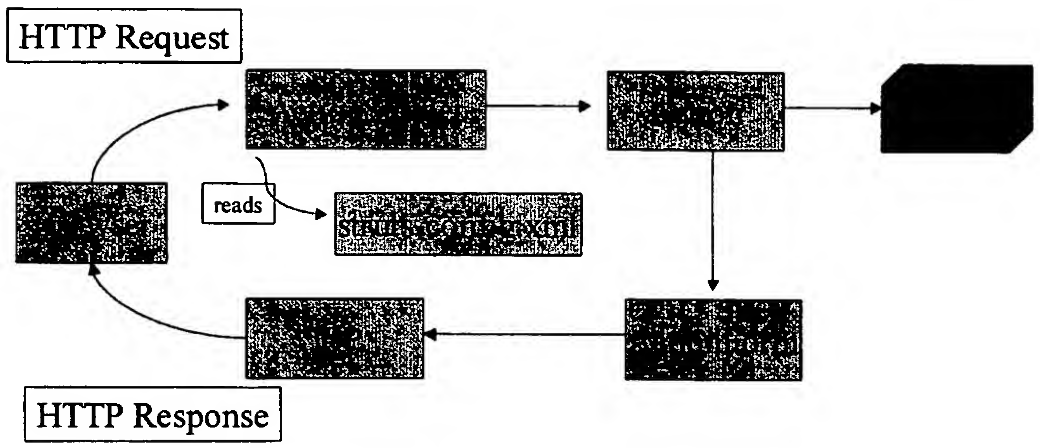


Figure 1: Struts Flow Diagram

When we build the application, we must first define the 3 part of the application and attach each one to the development process.

Struts Components

Controller – The ActionServlet is responsible for dispatching all requests. Mappings to actions are defined in the struts-config.xml file
Action – Actions are defined by extending the Action class. Actions perform actual work.
Model – The ActionForm can be extended to provide a place to store request and response data. It is a JavaBean (getters/settters)
View – JSP pages utilizing Struts tag libraries

Sequence diagram

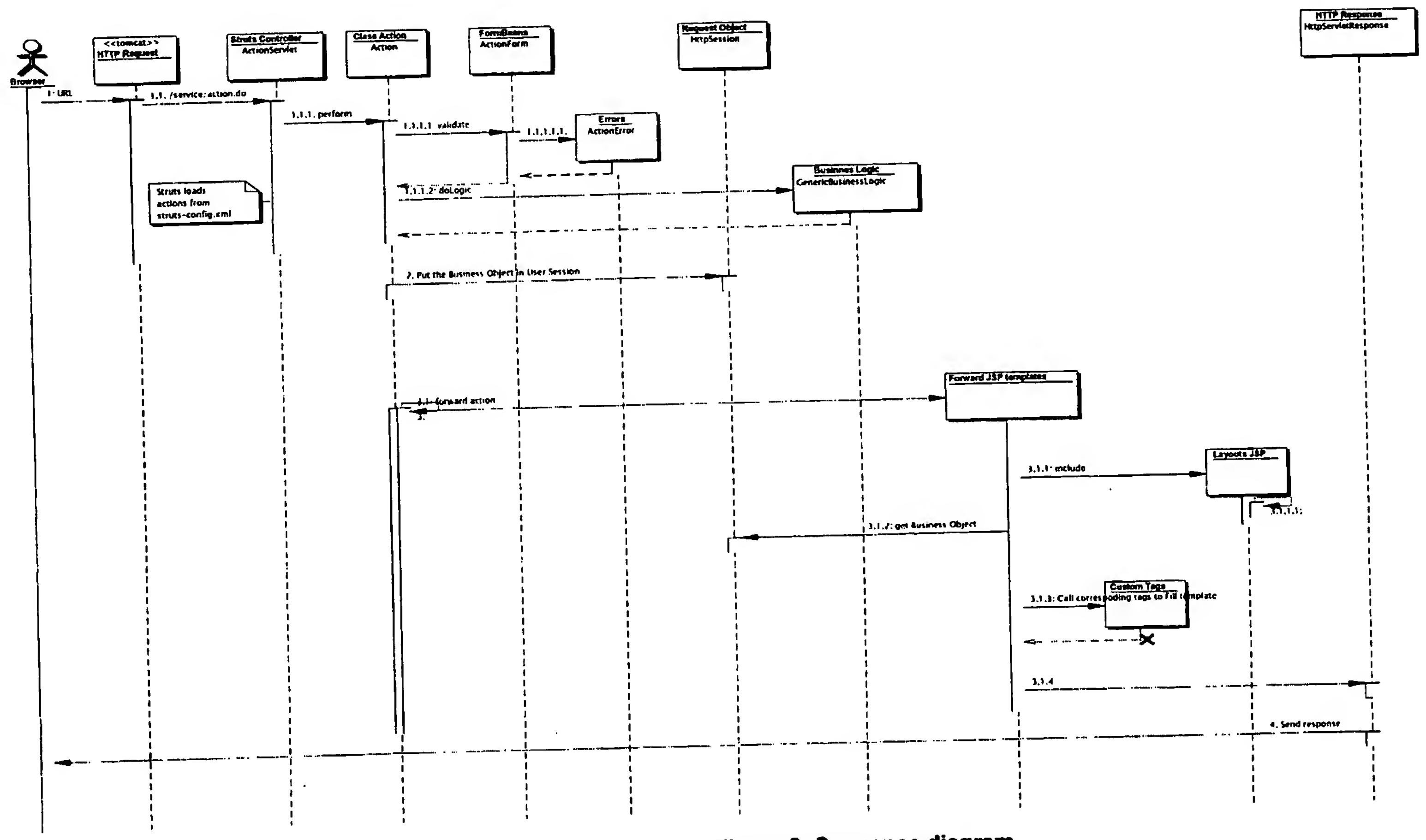


Figure 2: Sequence diagram

WEB SITE DESIGN

First, to start using Struts for the developm nt of a web application, you will need some requirements before starting any coding. These requirements are:

- The scenario with all possible click paths to get content
- The business logic use cases
- The mock-ups and visual page in html

The scenario

With the scenario, we will be able to define each action to perform (controller) in the site and what business logic (model) to initiate to show the result (view). As shown in figure 2, an action can perform, at the end, a forward to a view, usually a Java Server Page.

Sample Screen Flow

Welcome

- Create an account { Register }
- Login to an existing account { Login }

Register

Display form with registration properties; options to Save, Reset, and Cancel.

- Form: Username, Password, Password (confirm), Full Name, From Address, Reply To Address.
- Save: Validate registration properties; either return to user with advice, or store and display Menu
- Validations: Password and Password2 must match; From and Reply To Addresses must match email pattern. Reply To address is optional, all others must be non-blank and meet other validations. Username must also be unique to application.
- If succeeds: { Menu }
- If fails: { Register }
- Reset: Undo edits
- Cancel: Proceed to { Login. }

Login

- Display form to login user account, with options to Save or Reset.
- Form: Username, Password.
- Save: Validate login, both fields required, and must match stored registration (case-sensitive).
- Reset: Undo edits.

Menu

- Add/Edit subscriptions { Account }.
- Logout { Welcome }.

Account

- Display form to edit registration properties (see Register screen).
- List subscriptions in table; options to Edit or Delete { Subscription }.
- Columns must include Hostname, with other properties listed as space permits
- Option to Add (another) subscription { Subscription }.

Subscription - Add, Edit, Delete.

- Display form with subscription properties; options appropriate to task (Add, Edit, Delete).
- Options for Add, Edit Tasks: Save, Reset, Cancel.
- Options for Delete Task: Confirm, Cancel.
- Form: Mail Server, Mail Username, Mail Password, Mail Server Type.
- Save: Validate subscription properties; either return to user with advice, or store and display updated Account.
- Validations: Auto Connect may be blank (false), all other properties must be non-blank.
- If succeeds: { Account }.
- If fails: { Subscription }.
- Reset: Undo edits.
- Cancel: Return to { Account } screen.
- Confirm (deletion): Confirm: Delete record; display updated { Account. }.

Logout

- Invalidate user's login; return to Welcome screen.

Actions

Th actions are java classes and they must xtend the Struts Action class.
For example, to perform a search on the site, we will create a SearchAction to perform th business logic of the search. Be careful, the business logic is not coded in the action, but only called from it. In other words, a programmer that implement an action will just have to know how to create the business logic object and what parameters it needs. After that, it will put the objects in the session, application or request scope object for future use by the JSP pages.
In some cases, the business logic could be used to validate a form (ex: login, registration, etc ...). In this case, we can use the ActionForm classes to perform that.

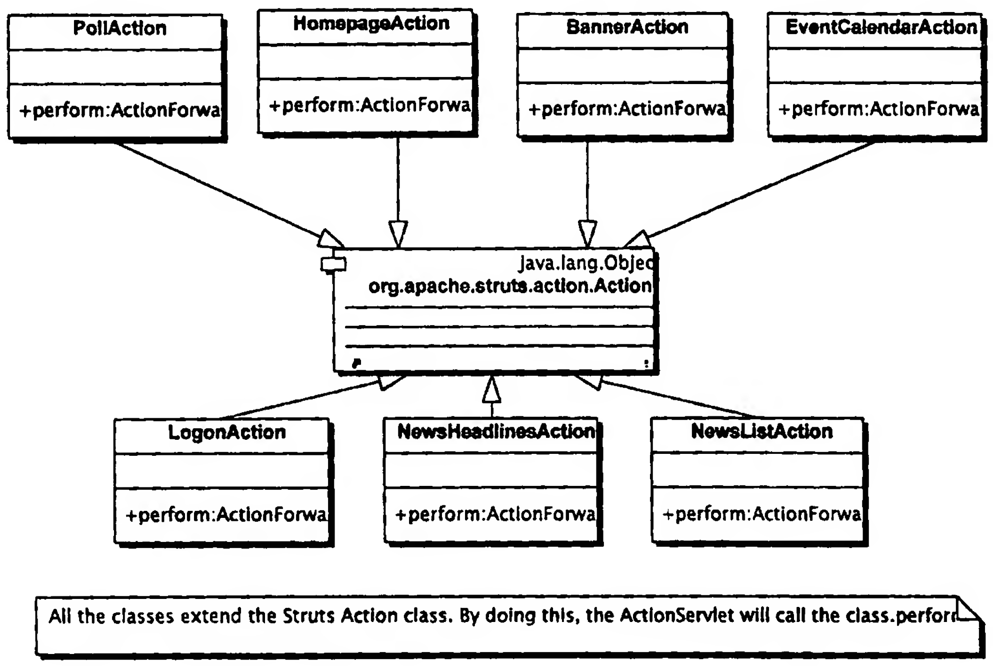
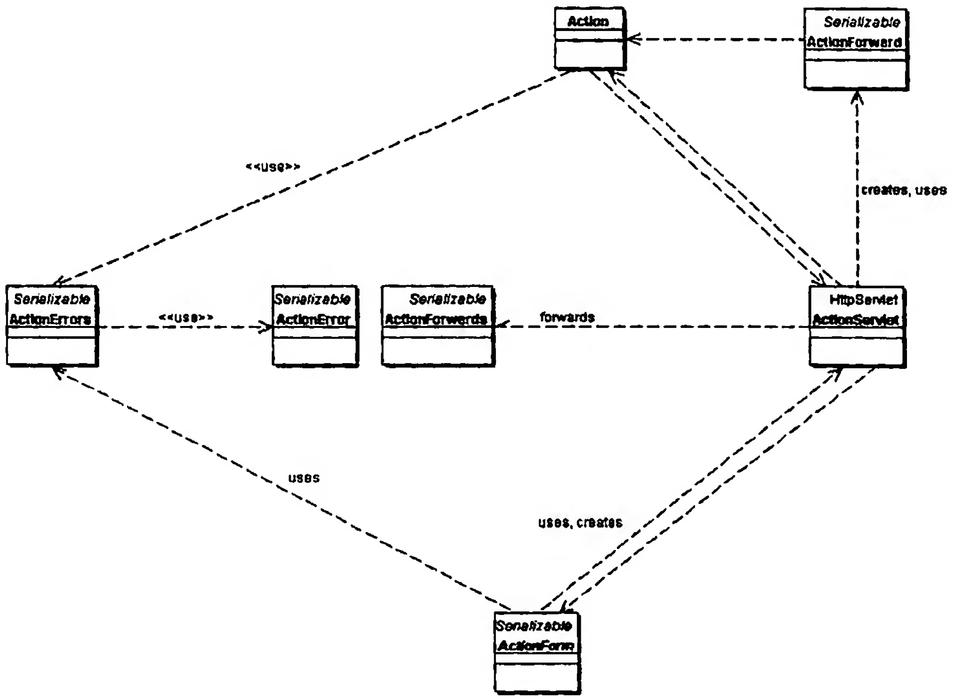


Figure 4: Action classes

Acti n mapping

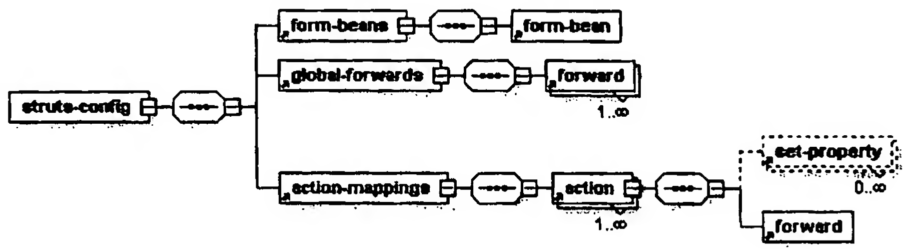
In the Struts framework, we can use a workflow stored in an XML config file, called *struts-config.xml*, to map each action to a class. In fact these actions are simple name finishing by a ***.do***. This names will be used by the ActionServlet to handles all HTTP request finishing by the extension *.do*. When the ActionServlet controller receives an action, it calls its appropriate mapping by loading its type.
This type is typically a class finishing by Action (ex: LoginAction), an include from an other file or an other mapping.
Each action can specify it's own CMS driver. This information can be sets in the Action Mapping properties.
By default an action mapping is provided with the Mediasurface CMS Driver but this one can be overridden later with another CMS Driver to get content from a Database for xample.



Sample Struts Action Mapping : struts-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE   struts-config   PUBLIC   "-//Apache   Software   Foundation//DTD   Struts
Configuration         1.0//EN"         "http://jakarta.apache.org/struts/dtds/struts-
config_1_0.dtd">
<struts-config>
  <!-- ========== Form Bean Definitions =================================== -->
  <form-beans type="org.apache.struts.action.ActionFormBean">
    <form-bean type="com.conceptis.actions.LoginForm" name="loginForm" />
  </form-beans>

  <!-- ========== Global Forward Definitions ============================== -->

  <global-forwards type="org.apache.struts.action.ActionForward">
  <forward name="login" path="/login.jsp" redirect="false" />
  <forward name="success" path="/homepage.jsp" redirect="false" />
  </global-forwards>

  <!-- ========== Action Mapping Definitions ============================== -->
  <action-mappings type="org.apache.struts.action.ActionMapping">
    <action   parameter=""   type="com.conceptis.actions.LoginAction"   path="/login"
name="loginForm" input="/login.jsp" scope="request">
      <forward name="success" redirect="false" path="/homepage.jsp" />
    </action>
    <action          parameter=""          unknown="false"          path="/homepage"
type="com.conceptis.actions.HomepageAction">
      <forward name="success" redirect="false" path="/homepage.jsp" />
    </action>
    <action parameter="" type="com.conceptis.actions.SideBarAction" path="/sidebar">
      <forward path="/sidebar.jsp" redirect="false" name="success" />
    </action>
  </action-mappings>

</struts-config>
```

Struts config Schema



Sample Web Application configuration for Struts : web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>debug</param-name>
      <param-value>2</param-value>
    </init-param>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>application</param-name>
      <param-value>ApplicationResources</param-value>
    </init-param>
    <init-param>
      <param-name>detail</param-name>
      <param-value>2</param-value>
    </init-param>
    <init-param>
      <param-name>validate</param-name>
      <param-value>true</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>database</servlet-name>
    <servlet-class>org.apache.struts.webapp.example.DatabaseServlet</servlet-
class>
    <init-param>
      <param-name>debug</param-name>
      <param-value>2</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>homepage.do</welcome-file>
  </welcome-file-list>
  <taglib>
    <taglib-uri>/WEB-INF/app.tld</taglib-uri>
    <taglib-location>/WEB-INF/app.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
  </taglib>
</web-app>
```

Forwards

To respond to the request, the action perform a the end of it's execution and when th business logic is done, it forwards to a view or an error page.

This view is normally templates that will thems lves load all it's content from the object stored by the action in th w bapps session, requ st or application objects.

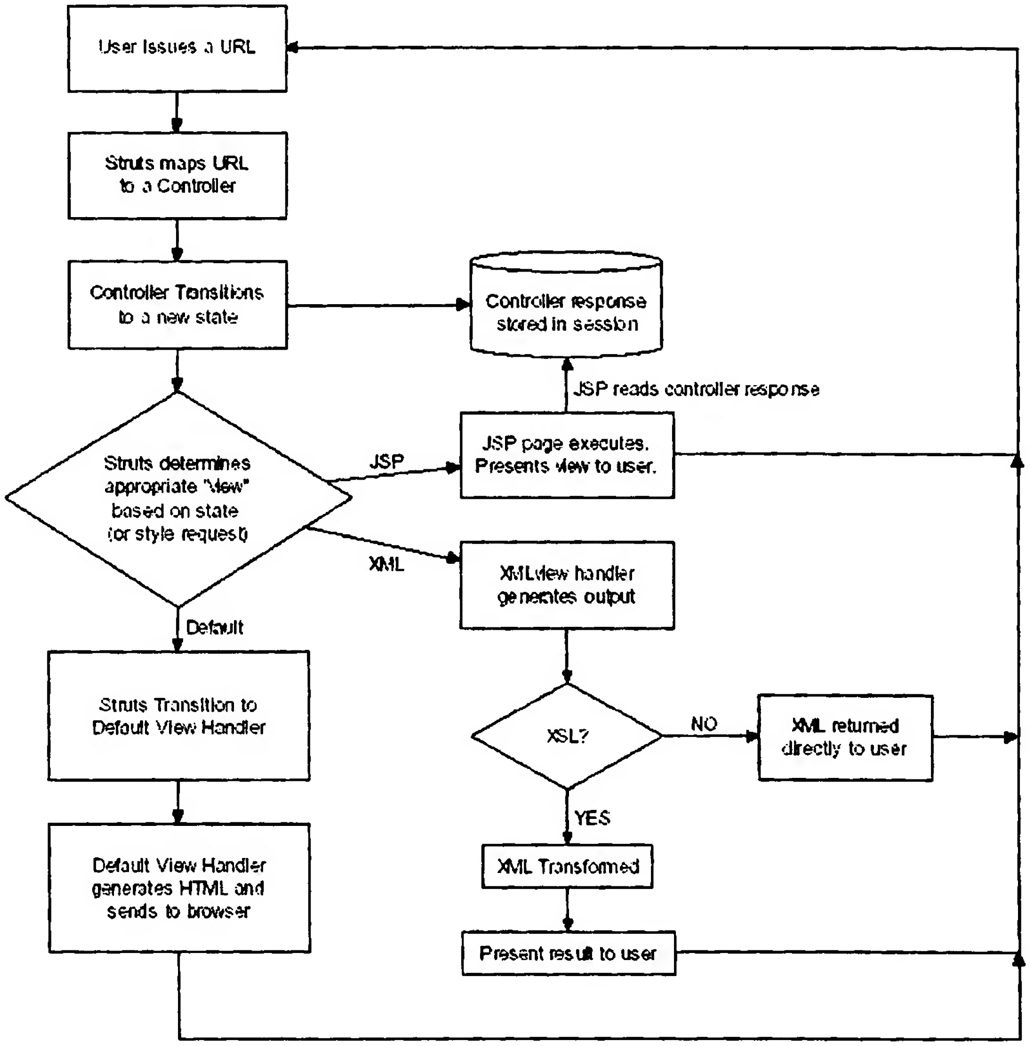

User Issues a URL
Struts maps URL to a Controller
Controller Transitions to a new state
Controller response stored in session
JSP reads controller response
Struts determines appropriate "View" based on state (or style request)
JSP
JSP page executes. Presents view to user.
XML
XMLview handler generates output
Default
Struts Transition to Default View Handler
XSL?
NO
XML returned directly to user
YES
XML Transformed
Default View Handler generates HTML and sends to browser
Present result to user

Figure 5: Forward flowchart

Business logic

The object to put in the session, request or application objects comes from various other classes.
To get content from different CMS, we will create a CMS Driver similar to JDBC Drivers to implement specific Document Type classes. For example, in the case of a News article we will create a getNewsArticle method that will be implemented by the Mediasurface package as well as any other content management system implementation package.

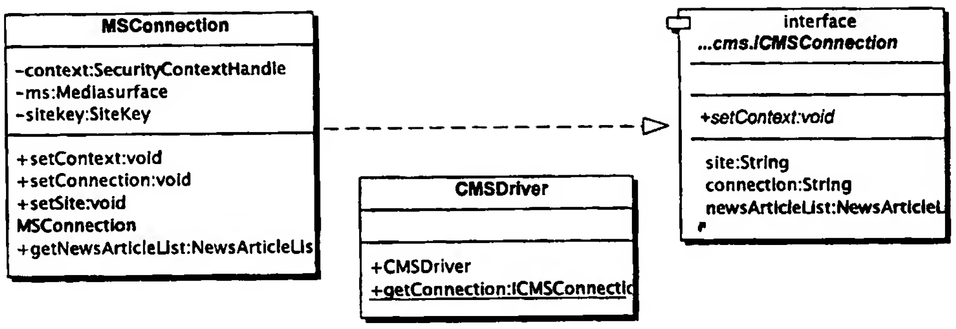


Figure 6: CMSDriver for Mediasurface

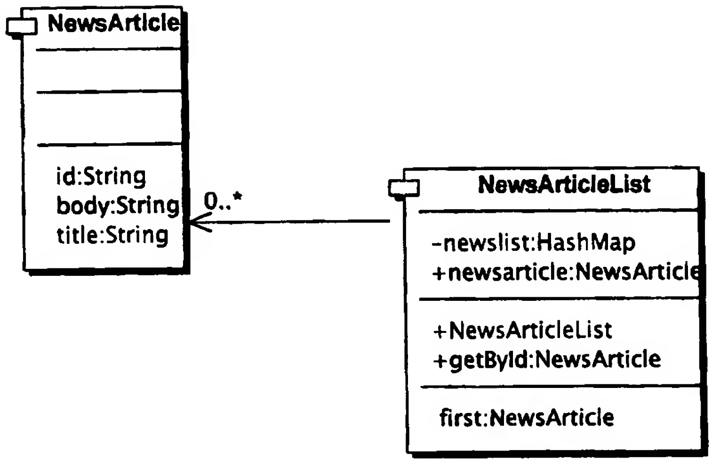


Figure 7: Sample Content Object

Security

The actions will be managed by the container security roles with a database (see Tomcat or Resin for implementation). To handle action with a security check, we must provide in the struts-config.xml a property describing the role.

```
<action path="/newsarticle" parameter=""
type="com.conceptis.contentdelivery.action.NewsArticleAction">
      <set-property property="Role" value="All" />
      <set-property property="Factory" value="default" />
       <forward name="edit" redirect="false" path="/NewsArticleEdit.jsp">
              <set-property property="Role" value="Administrators" />
              <set-property property="Factory" value=" default" />
              <set-property property="Role" value="Editor" /></forward>
       <forward name="list" redirect="false" path="/NewsArticleList.jsp">
       <set-property property="Role" value="All" /></forward>
</action>
```

Stats

Each action must provide a mechanism to store its state with a timestamp. This will be used to store stats in a common object created by a Servlet filter. As shown in the schema below, this filter will be responsible to serialize the log object in a Database.

This logging database could be read by an other application to put all this action log in the Data warehouse.

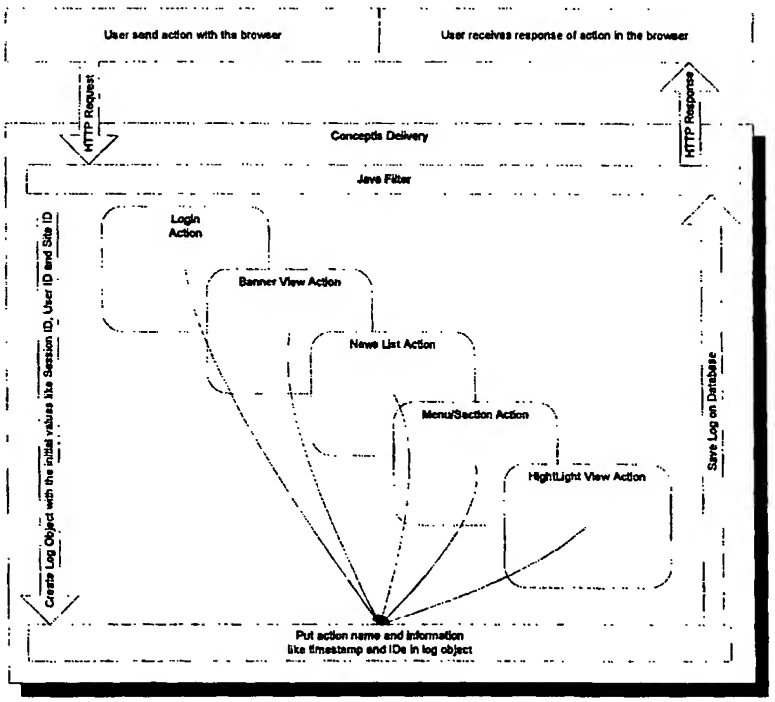

User send action with the browser
User receives response of action in the browser
HTTP Request
HTTP Response
Conceptis Delivery
Java Filter
Create Log Object with the initial values like Session ID, User ID and Site ID
Login Action
Banner View Action
News List Action
Menu/Section Action
HightLight View Action
Save Log on Database
Put action name and information like timestamp and IDs in log object

Figure 8: Delivery Architecture Overview for Logging Stats Actions (read from left to right)

Presentation

Templates

One of the most important think to understand is all about how this design solves the need in separating the presentation logic from the business logic. Because the template never instantiate object to get content but always getting it from it's scope objects (session, request or application).

When the action is forwarded to a view, this last called a JSP template to handle the content.
To have a very modular view, a layout can be used to present different views.

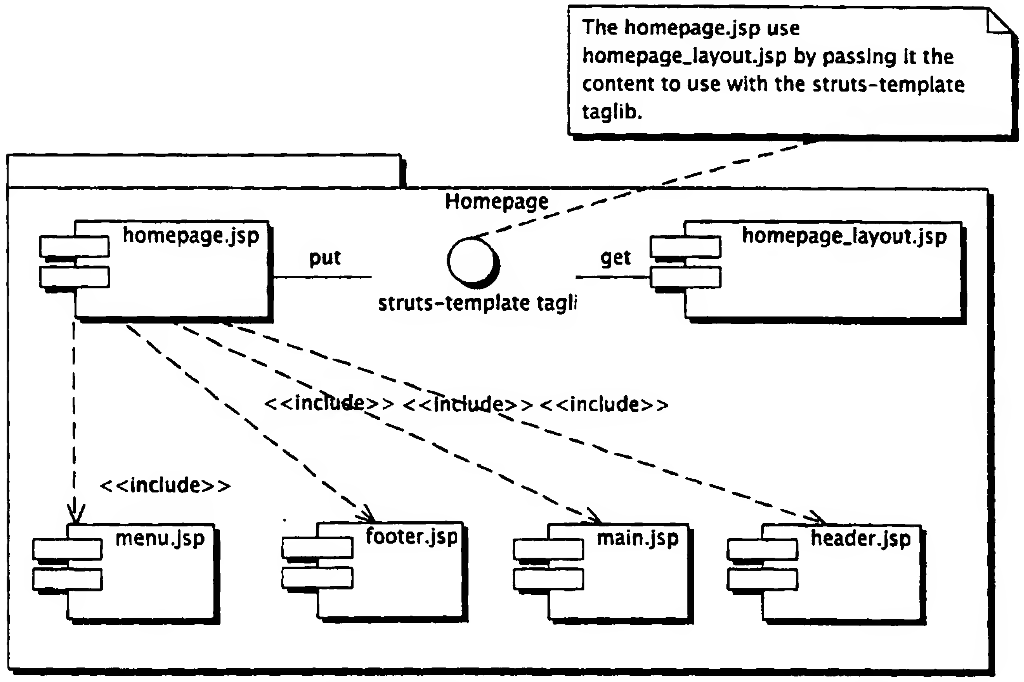


Figure 9: JSP Templates

Sample JSP Template code : homepage.jsp

```
<%@ taglib uri='/WEB-INF/struts-template.tld' prefix='template' %>

<template:insert template='homepage_layout.jsp'>
  <template:put name='title' content='Title' direct='true'/>
  <template:put name='header' content='header.jsp' />
  <template:put name='navigation' content='menu.jsp' />
  <template:put name='mainpage' content='main.jsp'/>
  <template:put name='footer' content='footer.jsp' />
</template:insert>
```

The template mechanism is provided from a Struts Taglib.

The taglib **'template'** use `homepage_layout.jsp` to present the content by passing it all the fields filled by simple visual beans (JSP or other visual tags).

These visual beans are simple JSP that encapsulate the entire logic to display a piece off content (banner, menu, poll, etc ...). Also, the object used to perform the business logic is instantiate only once and sets in the session. So after the first request, the objects can be reused to perform requests.

When using templates it's important to understand that the JSP included are not Actions but simple views without any knowledge of their context use. In other words, that's mean that a template calls JSP to perform a display and not an action. But this template could be called from an Action.

Sample Menu.jsp to show how to use a Visual Bean in a template : menu.jsp

```
<%@ page import="com.conceptis.logic.*"%>

<%
  SideBar sidebar;
  // check if the business object exists in session scope and get it
  if (session.getAttribute("sidebar") != null){
    sidebar = (SideBar)(session.getAttribute("sidebar"));
  }else // create a new one and set it in session
    sidebar = new SideBar();
    session.setAttribute("sidebar", sidebar);
  }
%>

    <html>
    <font size='5'><a name="top">Topics</a></font><p>
      <table width='145'>
        <tr>
          <td><a href='<%=sidebar.url%>'><%=sidebar.title%></a>
          </td>
        </tr>
      </table></p>
    </html>
```

Sample JSP Layout code : homepage_layout.jsp

The layout is filled from the result of each Visual Bean. It's here where we apply any look and feel and decoration.

We can also use a style sheet (css) to apply a transformation to the output (font, size, etc ...).

We can use the same template across the entire site or other sites and if a change is made here, the entire site will be affected at runtime.

Because business logic can change every time, don't forget to use taglibs as much as possible to get content so every change made in the business logic will not affect the presentation.

At last, templates can invoke other templates for a more complex document like CyberSessions of Slide Kits.

```
<%@ taglib uri='/WEB-INF/struts-template.tld' prefix='template' %>

<html>
<head>
<title><template:get name='title'/></title>
<link rel="stylesheet" href="css/homepage.css"
      charset="ISO-8859-1" type="text/css">
</head>
<body>

<table>
   <tr valign='top'>
      <td><template:get name='navigation'/></td>
      <td><table>
            <tr><td><template:get name='header'/></td></tr>
            <tr><td><template:get name='mainpage'/></td></tr>
            <tr><td><template:get name='footer'/></td></tr>
          </table>
      </td>
   </tr>
</table>
</body>
</html>
```

Taglibs (Custom tags)

Most of the content can be encapsulate in taglibs for more reusability. These custom tags will be created using the same logic as template. That means, they just fill the content from scope objects and return the control to the JSP page.
A simple use could be a check login tags at the beginning of the homepage.jsp :

```
<%@ taglib uri="/WEB-INF/app.tld" prefix="app" %>
<app:checkLogon/>
<%@ taglib uri='/WEB-INF/struts-template.tld' prefix='template' %>
<template:insert template='homepage_layout.jsp'>
  <template:put name='title' content='Title' direct='true'/>
  <template:put name='header' content='header.jsp' />
  <template:put name='navigation' content=menu.jsp' />
  <template:put name='mainpage' content='main.jsp'/>
  <template:put name='footer' content='footer.jsp' />
</template:insert>
```

Taglib Class : CheckLoginTag.java

```
package com.conceptis.logic;

import java.io.IOException;
import javax.servlet.http.HttpSession;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.TagSupport;
import org.apache.struts.action.Action;
import org.apache.struts.util.MessageResources;


/**
 * Check for a valid User logged on in the current session.  If there is no
 * such user, forward control to the logon page.
 * @version $Revision: 1.2 $ $Date: 2001/04/14 12:53:07 $
 */

public final class CheckLoginTag extends TagSupport {


    // --------------------------------------------------- Instance Variables
    /**
     * The key of the session-scope bean we look for.
     */
    private String name = "user";


    /**
     * The page to which we should forward for the user to log on.
     */
    private String page = "/login.jsp";
    // ----------------------------------------------------------- Properties

    /**
     * Return the bean name.
     */
    public String getName() {
     return (this.name);

    }


    /**
     * Set the bean name.
     *
     * @param name The new bean name
     */
    public void setName(String name) {

     this.name = name;
```

```
}


/**
 * Return the forward page.
 */
public String getPage() {

return (this.page);

}


/**
 * Set the forward page.
 *
 * @param page The new forward page
 */
public void setPage(String page) {

this.page = page;

}


// ------------------------------------------------------- Public Methods


/**
 * Defer our checking until the end of this tag is encountered.
 *
 * @exception JspException if a JSP exception has occurred
 */
public int doStartTag() throws JspException {

return (SKIP_BODY);

}


/**
 * Perform our logged-in user check by looking for the existence of
 * a session scope bean under the specified name.  If this bean is not
 * present, control is forwarded to the specified logon page.
 *
 * @exception JspException if a JSP exception has occurred
 */
public int doEndTag() throws JspException {

// Is there a valid user logged on?
boolean valid = false;
HttpSession session = pageContext.getSession();
if ((session != null) && (session.getAttribute(name) != null))
    valid = true;

// Forward control based on the results
if (valid)
    return (EVAL_PAGE);
else {
    try {
        pageContext.forward(page);
    } catch (Exception e) {
        throw new JspException(e.toString());
    }
    return (SKIP_PAGE);
}

}


/**
 * Release any acquired resources.
 */
public void release() {
        super.release();
        this.name = "user";
        this.page = "/login.jsp";

    }


}
```

STANDARDS

Struts configuration

An application must provide a `struts-config.xml` that contains all possible maps and actions for this site. These actions could be specific to the site and will be added to a generic config file. This generic file will contain also global forwards that could be used by any site.
So when we deploy a site that's means we must provide a struts-config file, a list of new JSP and a jar with its specific action classes or business logic.
The shared library that contains global action classes and the drivers is in the classpath of each instance of the web servers and it's located in the same place for all.
The name of the actions is a part of the name of the document type. For example, a NewsArticle type will have an Action called *NewsArticleAction* and its map will be named as its functionalities, for example: **edit, view, list, etc.**
With this convention we will be able to determine witch object to get with the driver.

As the new Struts version 1.1 will gave some utilities classes to get an action config, we will be able to put custom properties for actions that use global parameters in the struts-config. Like for example a login Action that can handle the user's default security levels.

But for now, we must read the struts-config file and setting a common object in the application scope and it will contain all the custom properties like a *properties* file.
By mapping actions name and document types, we can implement a generic library to manipulate document in general manner. This library will use the CMS driver to get content. It must also provide a pool of object if necessary as well as the driver must provide a pool of connection for different CMS user (Mediasurface).
Now to use the struts-config as the main config file for every web site we must define the request language to call these actions as this:

A request is an HTTP request format and must be created as this:

```
/<ActionController>/<Action name>.do? <mapping=Mapping name> & [& param1=value & param2=value & ... paramX=value]
```

Where:

ActionController:	the name of a controller that will handle actions for a service.
mapping:	the name to witch the action has to map if succeful.

For example the News Article request will be called like this:
`NewsArticle.do?` **`Mapping=List`** `& group=MD & from=1 & to=25`

Keep in mind that we must use a controller to handle HTTP request only like a Search service or Syndicate Data in XML format.

In general, the `mapping` parameter will be used by the action to know what to do if succeful business logic is done. If not, it has to map to *mappingvalue*`Error`. Where *mappingvalue* is the `mapping` parameter value.
In conclusion, the struts-config will provide all the actions and there classes, the mappings and the custom properties for the entire site. In fact, this file could be used to define all th workflow of and could be us d as a template for other sit s.

Packages

They are several packages to use when developing a new site. The most important thing to remember is to never instantiate any Mediasurface class from Action class or JSP/Taglib. When there is a need to get a business object always go throw Conceptis API witch provide simple classes to get object from MS or any other services.

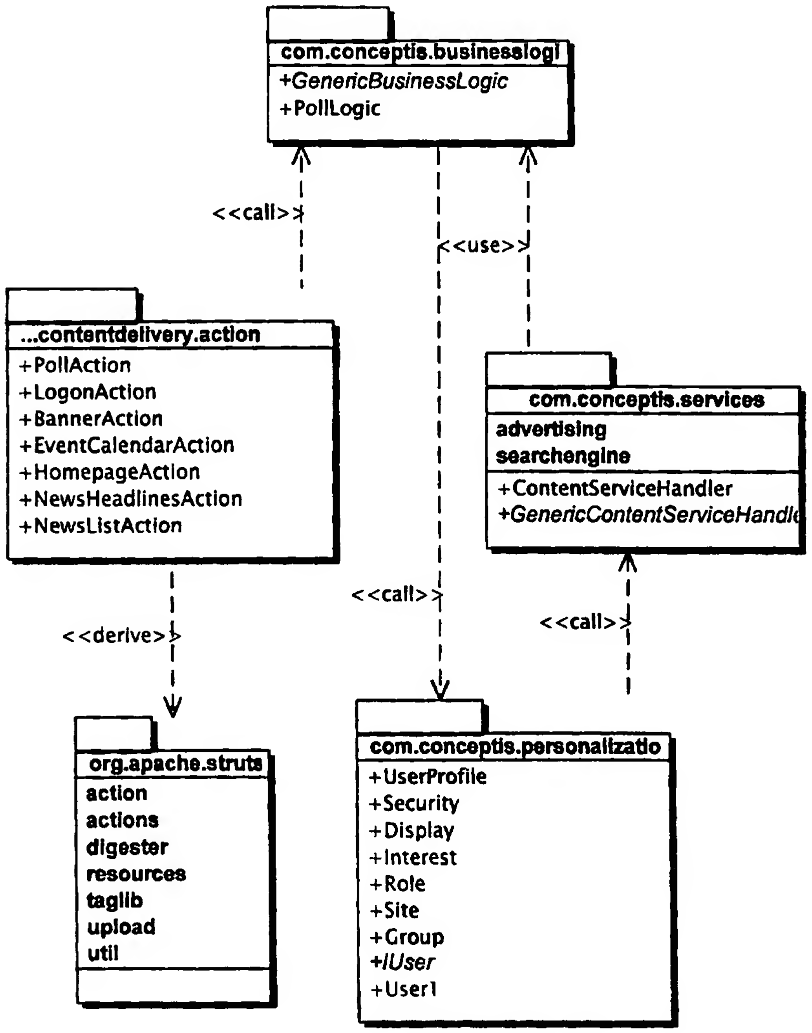


Figure 10: Top Level packages

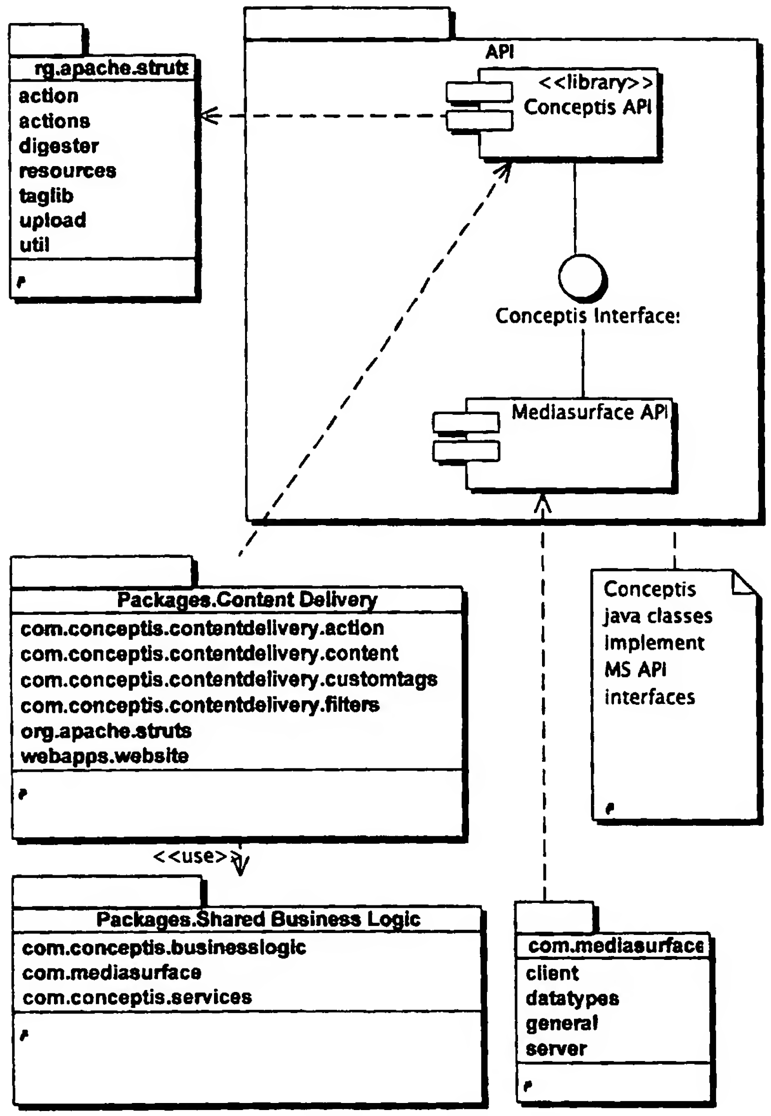

rg.apache.struts
action
actions
digester
resources
taglib
upload
util
API
<<library>>
Conceptis API
Conceptis Interface:
Mediasurface API
Conceptis java classes implement MS API interfaces
Packages.Content Delivery
com.conceptis.contentdelivery.action
com.conceptis.contentdelivery.content
com.conceptis.contentdelivery.customtags
com.conceptis.contentdelivery.filters
org.apache.struts
webapps.website
<<use>>
Packages.Shared Business Logic
com.conceptis.businesslogic
com.mediasurface
com.conceptis.services
com.mediasurface
client
datatypes
general
server

Figure 11: Components view

URI

To handle web request we must provide different URLs. Wh n these URLs are used to perform an action from a web link, we must generate this URL with an action mapping. We should never create a link with a .jsp or other extensions to perform a request because Actions must call the business logic to decide witch view to forward.
This URL format must be unique to help a Proxy Cache to identify unique request and manage them easier.

Folders

A web application is a set of files and libraries that are assembled in zip file with the extension .war

```
└──website
    ├──css
    ├──graphics
    ├──jsp
    │   ├──includes
    │   └──templates
    ├──html
    └──WEB-INF
        ├──classes
        │   └─com
        │     ├──conceptis
        │     │   ├──businesslogic
        │     │   ├──contentdelivery
        │     │   │   ├──action
        │     │   │   ├──content
        │     │   │   ├──customtags
        │     │   │   └──filters
        │     │   ├──contentmanagement
        │     │   ├──personalization
        │     │   └──services
        │     └─mediasurface
        │        ├──client
        │        ├──datatypes
        │        ├──general
        │        └──server
        │            ├──admin
        │            └──util
        └──lib
```

`WEB-INF` is the main folder where to put web site custom classes and libraries.
`JSP` is the main folder to put JSP Includes and Templates.
`HTML` is the main repository for static document like Helps.
`CSS` is the main folder for Cascading Style Sheets.
`GRAPHICS` for all images.

Internationalization

The Struts framework offers the possibility to put web captions lik button names or rrors messages in a multiples files called properties files. These properties files can be shared across all the web sites and manager by changing the name with an ISO country abbreviation.
In Conceptis we will use a file by site and language. For example we will create a file `tho_en.properties` for thehart.org captions in English.
These files will be loaded by the web application at startup and Struts will manage them by checking the web browser language.

Deployment

To deploy a web site we must use Ant to compile classes before sending them to the desired server (dev, test, prod).
In the deployment process, there is always a set of files that must be checkout from CVS repository.
See the Conceptis Deployment Process for more information (ref #7).

CONCLUSION

By using this framework approach we will gain a lot in reusability of development but also in analysis for new web sites. The Struts Framework provides us a lot concepts that programmer usually thinks about every time they create a site and by using MVC pattern they will focus more on creating library of code instead of library of sites.

ANNEX 3

CONCEPTISTECHNOLOGIES DESIGN REVIEW SUMMARY

Revision of the John's Document

Table of contents

1. INTRODUCTION 111
1.1 HISTORY OF REVISION 111
1.2 Terminology 111
1.3 References 111
1.4 Assumptions 111
2. ISSUES 113
2.1 Content Structure 113
2.2 Types 114
2.3 Surveys, Polls and Forums 114
2.4 Images 116
2.5 Struts Framework 116
2.6 Oracle Support 116
2.7 Performance 117
2.8 Reporting 117
2.9 Delivery 117
2.10 Searching 118
2.11 Data Transfer 119
2.12 Existing Data 120
2.13 W rkflow 121

2.14 Personalisation 121

2.15 Future Utilities 121

INTRODUCTION

HISTORY OF REVISION

Revised	Auteur	Changes descriptions
17/07/2002	Eric H	Add comments on the John's Document

Terminology

References

[11]

[12]

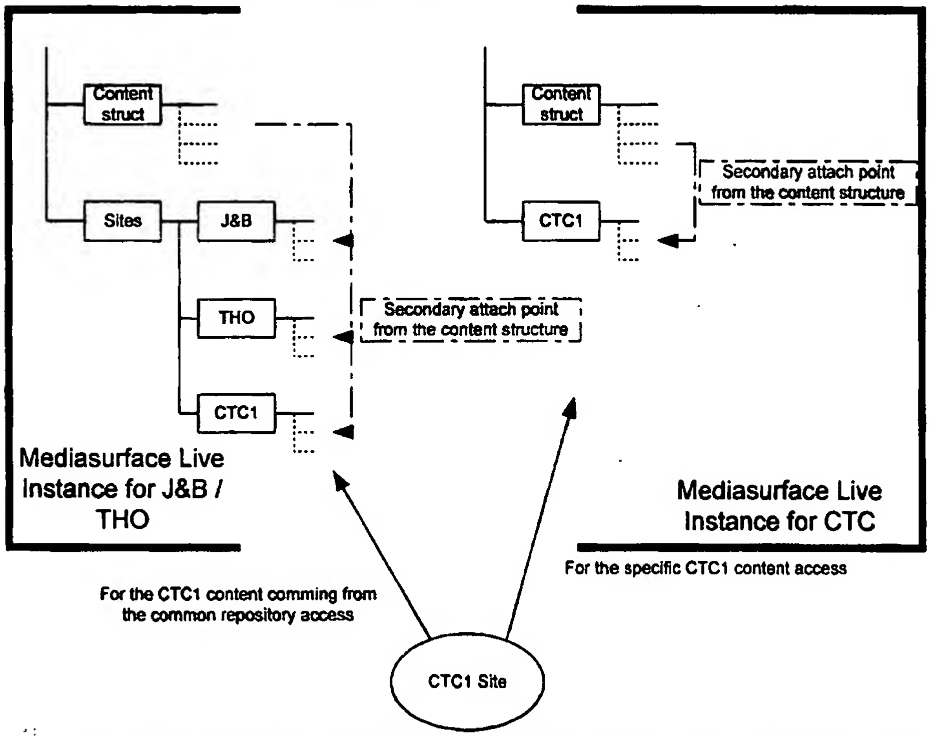
[13]

Assumptions

Issues

Content Structure

It was agreed that 'Joint & Bone' and 'The Heart' websites would store their data within one repository to facilitate content and type sharing. Each site would have its own branch off the root. Other sites that have to have non-sharable data for legal reasons, such as the CTC sites, will use a separate repository. The J&B/TH repository will have branch for secondarily attaching items to be shared with the CTC and other external sites. In these cases a repository is equivalent to a Mediasurface site which can be split out into a different Oracle user if required.


Content struct
Sites
J&B
THO
CTC1
Secondary attach point from the content structure
Mediasurface Live Instance for J&B / THO
Content struct
CTC1
Secondary attach point from the content structure
Mediasurface Live Instance for CTC
For the CTC1 content comming from the common repository access
For the specific CTC1 content access
CTC1 Site

The CTC1 Site will access two Mediasurface Live instances. One for his own content, and the other for the shared content coming from the main content repository used for THO, J&B...
There is a preference from Conceptis to have all items and Branch types. It was recommended that typical leaf type items such as News Articles would be more manageable and performant as leaf types.(In our case, we will use Branch Type for all document types. I.e. a news article Item will have a word document directly attach to him). As searching is done outside Mediasurface (apart from Reporting), items are normally referenced directly via their itemid and there is not a requirement to navigate the site tree often, the final conclusion was that these advantages were outweighed by having the flexibility to use attachpoints on all items.

Types

The following points were clarified:

- It is not possible to constrain access on a per-user basis for specific fields in an Item definition.
  In this case we will use a XML field defining the displayable and access right of items fields.

- Navigation can be constrained through collections and types but not views in the instance because Conceptis are not using the Mediasurface Delivery Framework.
  We will use the notion of role on the delivery side to add a level of access on action a user can performed.

- A generic download for picture can be achieved by using any multimedia image type as Mediasurface treats them all in the same way.

- If an image is not a relation it is still possible to retrieve it by its url, path or id. The id is the safest option as it will never change. All Images, or multimedia files will have a dedicated Type and a primary attach point on the content structure. Items will refer them using the XMLString. In general all multimedia Items will have one and only one attach point (the primary to there dedicate Content structure section)

- String fields with formatting are stored as HTML.

- It is not possible to add and attribute to a field short of including XML or some other parsable format within the field.
  The delivery wills pars the fields to determine if it contains XML or not.

- It is possible to know how many times and item is requested by using the logs although this is not in a format friendly to generating reports.

Surveys, Polls and Forums

It was agreed that in the case of surveys and polls, where aggregation of results is the main aim, that the questions would be stored in Mediasurface but the results would be written to an external table. As there is no Content Management in terms of results, this streamlined approach is by far the most efficient approach. After analysing the requirements, it was agreed that M diasurface could b used to facilitate Forums. A different ItemType will be used for the Forums Section, Topic and Message.

In other word:

For Forum, topics, and messages, they will be stored in Mediasurface directly in the site section.

For Pool, survey, quiz, the message or questions (envelop) is store in Mediasurface. The answers are part of the user profile, or related sponsore.

Images

Rather than referencing images in an Item field with the necessary data about page positioning, it was decided that it would be more efficient to use Mediasurface Inlines functionality. This is because the image metadata about page positioning would better belong in the page hence keeping the layout and the content separate. As this is the only metadata required for an image there is therefore no need to use an item field to reference the image. The further advantage would be performance as each image request would not require an XML parse before being presented on the page. Therefore inlines was decided as the preferred solution with all images being attached to a central image repository.
We will used the XMLString to refer to images. The xml relation will define the section ("position") where the image will take place.

Struts Framework

The issue of the best way of delivering the Mediasurface functionality within a Struts framework was analysed and a change from the existing design was suggested. The suggestion was not to have a separate data bean per Item Type but to have a generic Type data bean with the required type being specified as a parameter within the action bean. This is a more elegant approach and saves the maintenance overhead of having to alter each specific Type data bean each time there is a change to the corresponding Type specification. Although Struts does not allow the required HashMap datatype to handle Types as a parameter in the action bean, it was found that a simple modification to Struts would fix this.

Oracle Support

Mediasurface 4.0 is only supported on 9.0.1. Conceptis want to use later versions to take advantage of the most recent bug fixes. It is recommended that Conceptis talk to Mediasurface Support to come to some arrangement regarding their support contract, as using non-supported versions will cause invalidation.
(See Daniel for further information)

P rf rmance

Each individual client of the MAe does not pool a set of connections to the MAe, whilst this may not cause a bottleneck it was concluded that Conceptis should write their own connection pool for connections to the MAe from the Conceptis API to ensure that no bottle-neck occurs as it will not take long to implement and eliminates one potential performance issue. (The pooling is implemented. See in annexe the Adam R performance document "Performance Comparaison.doc") The MAe pools its database connections which can be set in the properties file, normally about 10 connections will offer the best balance between throughput and number of connections. The only other pooling is done in thread pools which are not implementation dependant, these include the scheduler and the stream handler, as well as the implicit RMI server thread pool.
The caching processes of the MAe were discussed and Conceptis learned of how the MAe will flush an items cache whenever it is updated proving there is only one instance of the MAe running or syncd is running. Also, it was explained that different expiry times can be set for the different elements of Mediasurface. For example, items can expire at a different time length to types etc. It was agreed that no extra optimisations on MAe caching should be performed unless it is perceived as a bottleneck later on in the project. If this is the case later on then the cache allocation can be increased and JMS could be used to implement pre-emptive caching. To scale for higher traffic it was recommended that clustering the app server would be the key facilitator. (From now the clustering is not the priority since our bottleneck is the MAE access and item retrieving. The performances issues we face for the moment is due to the 'long' time spend on a request to Mediasurface to retrieve an Item. In addition, we make a lot of recursive call to MS (for the menu or the forum). The solution, for the previous point, is to use a kind of site tree object specific for each MS user. This site tree can be part of a pool of object and a low-level priority thread can run on back ground to refresh this object every x Minutes/seconds)

Reporting

Conceptis were informed that the java client Reporting Tool uses a different api that is inaccessible via the MAe. This is true of all java client functions. To create a reporting tool using the available MAe it is recommended that Conceptis use the Advanced Search Functionality. The best place to start learning about this is the itemSearch class. (From now the content department is award that we will use the MS Java Client reporting tool for the first phases of the project). We will used an external search to perform filtering and search on the CAL. This will help us to create in a second phases a reporting tool.

Delivery

Conceptis were informed that it is possible to specify an order when requesting items using bindings or searching as they are always returned as an *itemList* object which allows sorting by item attributes and user defined fields. Conceptis were warned to make use of this and not to sort twice, once with the MAe and again with the CAPI, as there would be an effect on performance.

The issue of private and public cache time was deemed irrelevant because Conceptis are not using the Mediasurface Delivery Framework. In the Type definition the Cash times is set to 0.

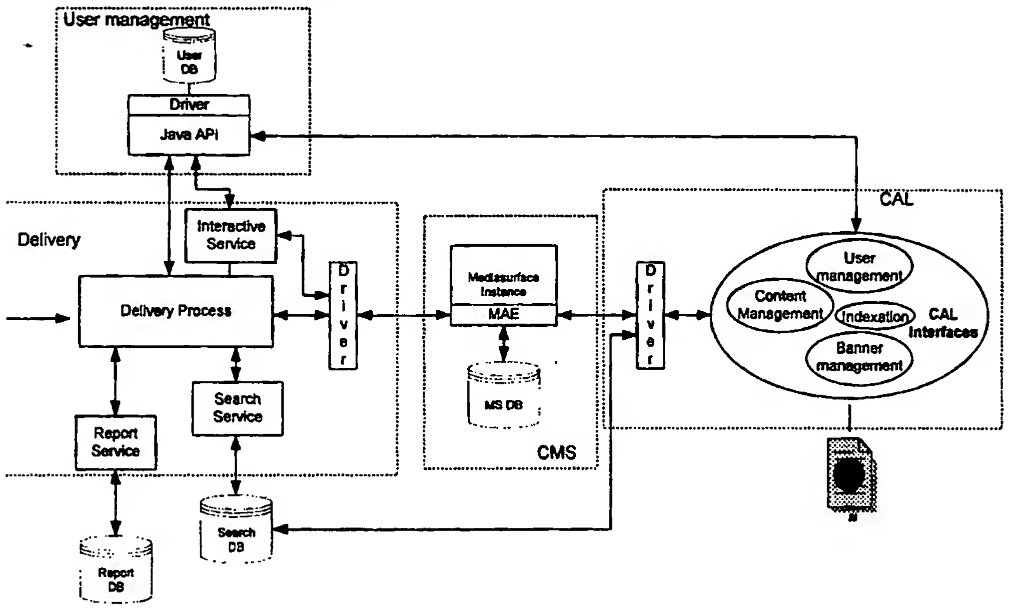
When building an Edit screen it was recommended that Conceptis store all the changes in the session until the user is ready to commit. This will allow the user to work over a number of pages, and to preview and/or cancel changes before finally committing. The item can be locked on the first edit screen by calling item.edit(), after the data has been set item.abandonChanges() or item.saveChanges() can be called. See the IEditable Interface for reference. It may be the case that other changes may be being stored and committed that are destined for a different database than the Mediasurface repository, especially in the case of the CAL. The Conceptis API will have to filter those elements to a different commit/abandon method and it may be more straightforward to use a different stored object on the session. This will be done during the CAL development.

Searching

When Mediasurface items are updated through the Conceptis API it was decided that searchable properties of the item along with the itemid would be written to an external database in XML format. This is so the data can be indexed and optimised for searching and so the site specific searches won't have to search through the content of all sites – this would be the case if searching was done within Mediasurface as a requirement from the Content Staff is to have all content in one site repository. There is also the case where a very small site may want to access a small subset of shared content; in this case by extracting the shared content into its own table this kind of search is also optimised. If other content independent of Mediasurface was to be added using the CAPI this would also mean that this data could easily be integrated for searching. It is this kind of flexibility that made this approach preferable from using the Content Gateway to export the Content. (We will not use the content Gateway to export the content due to a miss of information. An export XML module will be added on the driver (Marc D) to serialize Item in xml and export it in an Oracle Intermedia table. The search engine will used this table to perform site user and Content user search)

There is a potential performance issue with this approach. What happens is that when an itemid is returned in the search results then it is at some point accessed via the MAe if the user wishes to view that item, and it is returned to the user as long as the user has access to that item. If the requirement is to present the user with only items in the results that the user actually has access to then all the items will need to retrieved beforehand to check the access. (When ID's are returned, the delivery will ask MS through the driver to retrieve this Items using a specific MS User connection. If the User has no access right to this Item, this will generate permission Error. This error can be easily catch and handle)

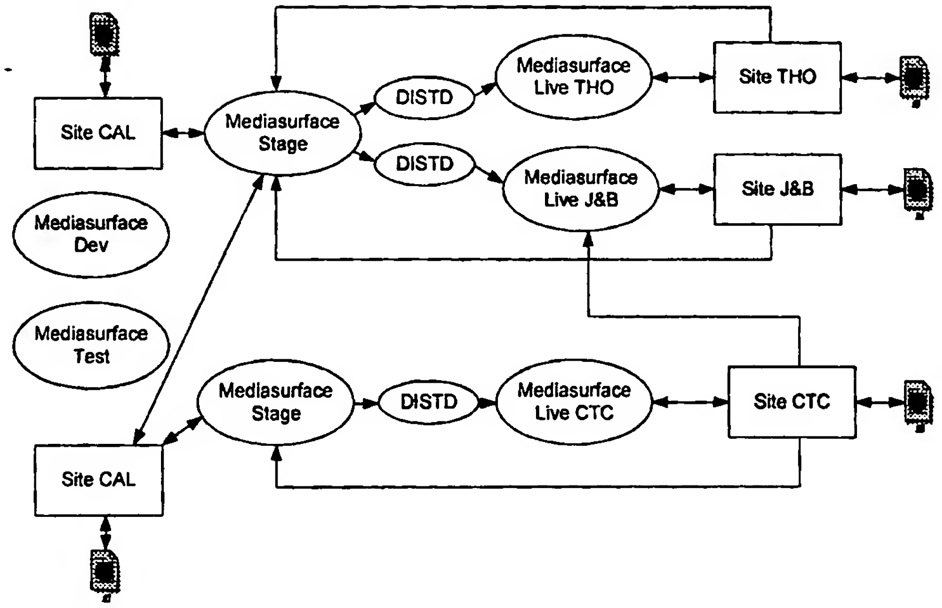
Overall Architecture.



Data Transfer

It is recommended that Distd is not used to filter on types as this is likely to cause problems with master/slave dependencies. Conceptis will just filter on status and maybe collections. Distd will be used to push content from the TH/J&B staging server to the TH/J&B live servers and also the CTC staging servers to the CTC live servers. There will normally be more than one live server to facilitate failover and loadbalancing.

For ach repository there will be only one Staging server as this acts as the master. The live servers are read-only whereas the staging is read and write. In the case of the CAL and the adding of forums, a connection to the Staging server will have to be made. As well as loadbalancing and failover, the other advantages to this setup include data security and keeping the processing required for editing items separate from that required for delivery so the performance of each is improved.



Using this configuration (for THO and J&B Live, and two DISTD), DISTD will have the role of clustering.

As well as the production environment there will be the development and test environment. For each Mediasurface development and test instance there will also need to be development and test instances for Conceptis specific data. The test Conceptis instance will be imported from the Conceptis development instance. In Mediasurface the test instance will be exported from Staging after the necessary changes are made on Staging. Once it has been proved that the test Mediasurface Staging instance works with the Conceptis test environment then the Conceptis test is moved on to work with the actual Staging environment.

Existing Data

It was decided that for the import of existing content from the old Conceptis system that th Integrator would be used. All the content exists in an XML format that can be transformed to be used with the Integrator. The main issue is how to map existing relations and inlines. Relations will be stored in XML in Mediasurface Item fields because Mediasurface relation functionality cannot map the whole required relationship sp cification; particularly essential is the classification of relations. This means that after the items have been committed, the relations will need to be then mapped from old to new id's using a custom script. A similar mapping will also need to occur for adding inlines.
After some evaluation the Integrator is not a valid solution (to heavy, not modular, long time process. To perform the Migration of existing data, we will used the drivers and cal functionality. Thom B will define a migration plan using Farid as resources.)

W rkflow

- When signing off a new item in workflow the itemid will change. This adds an extra level of complexity because the itemid is refer nced in searching and in fields holding relation information in XML. The solution is for the CAPI to update the itemid in these places if there is a new itemid on signoff. This is easily checked as the signoff method on IItem returns the new item from which the new itemid can be retrieved and compared to the old one. Because of deadlines Conceptis may not sign off to a new item for the first release.
(True, we will used the same version during all the life cycle of the Items (from 1.0 to 1.99). The major number will not change, the minor number will go from 1 to 99 (1.xx))

Personalisation

There was a discussion about how to best implement the personalisation of Conceptis pages. Conceptis personalisation works only on the elements of a single page. This means that item fields can be used to store data relevant to personalisation without causing performance issues. When a page is called the personalisation metadata field will be retrieved and the CAPI will perform the necessary logic that maps conditions between this data and user data from the Conceptis User Management System. This part will need a further analysis in collaboration with Eric G (for the User personalization part), Hugo T (for the display and action) and Sam G.

Future Utilities

There was a discussion regarding the possibility of Mediasurface providing scripts in the future for importing types and printing reports on type properties. Whilst the former would not be produced in time even if Mediasurface embarked on the project now, the latter script would be useful to Conceptis and so they will be informed if any dev lopments regarding this go ahead.

ANNEX 4

DEFINITION DOCUMENT

Table of contents

1. INTRODUCTION 122

1.1 HISTORY OF REVISION 122

1.2 Terminology 122

1.3 References 122

1.4 Assumptions 122

2. MEDIASURFACE 123

2.1 MediaSurface Overview 123

2.2 MediaSurface Interest 123

3. CONCEPTIS TECHNOLOGIES 127

3.1 History 127

3.2 Conceptis Technologies' requirements 127

4. INTEGRATION OF MEDIASURFACE INTO CONCEPTIS TECHNOLOGIES' WORLD 128

4.1 Defining content 129

4.2 Acquiring and storing content 130

4.3 Managing Users 133

4.4 Delivery 134

4.5 CAL: all-in-one website for Mediasurface workers 139

4.6 Surrounding Module 143

4.7 Hard Ware 144

5. TASK DEFINITION AND RESPONSIBILITY 149

6. ACTION PLAN 152

6.1 Action 152

6.2 Time 155

7. TIME LINE / RESOURCES 157

7.1 Resources estimate for the content and development team. 157

7.2 Time Line 157

8. ANNEX 161

8.1 Delivery Architecture 161

8.2 CME/CMC 162

8.3 MVC Introduction 164

INTRODUCTION

This document will d fine th global overview of the Mediasurface Project. This project can be define like this:

Manage and deliver content and sites from a common repository.

This repository is called Mediasurface.

In addition of the content management work, all the delivery mechanism using Mediasurface is to redefine since this is using the Java technology. This will give us the ability to create a modular, manageable and reusable solution to deliver content crossover all ConceptisTechnologies site.

HISTORY OF REVISION

Revised	Auteur	Changes descriptions
18-02-2002	Eric H	First Draft
26-02-2002	Eric H	Second Draft
26-03-2002	Eric H	Add Content
13-05-2002	Eric H / Tom H	Add Content
13-05-2002	Tom H	Revision / Add Content

Terminology

MS	MediaSurface
WF	Workflow
MAE	MediaSurface Application Engine
CMC	Content Management Console
CME	Content Management Engine
CAL	Conceptis Administration Layer

References

[14] MediaSurface Technical Overview

[15]

[16]

Assumptions

MediaSurface

MediaSurface Overview

Briefly, MediaSurface (MS) is a Content Management System (CMS) with the facility to display information to web-based end-users. Because it was first designed from an ***editorial*** point of view, MS has extensive facilities for managing the flow of content, the versioning of documents, the mapping of relation between content, and so on.
The display of content to end-users, both in-house and internet-based, is done through a Java API. Thus web browsers have access — via an embedded security layer if desir d — to all the content held in the Mediasurface repository.
In technical language, "MS 4.0 uses Oracle 9i to store the content. For the delivery side, the library and servlets are supported on Tomcat 4.0.1 and BEA WebLogic 6.1, and are written against version 1.1 of the Java Server Pages specification and version 2.2 of the Java Servlet specification. The default mark-up produced by the tags is HTML 4.01."
A good overview document from the MediaSurface supplier may be found at:
\\Web4-mtl\intra\htdocs\mediasurface\MediaSurface_4.0\Mediasurface 4.0 Technical Overview.pdf

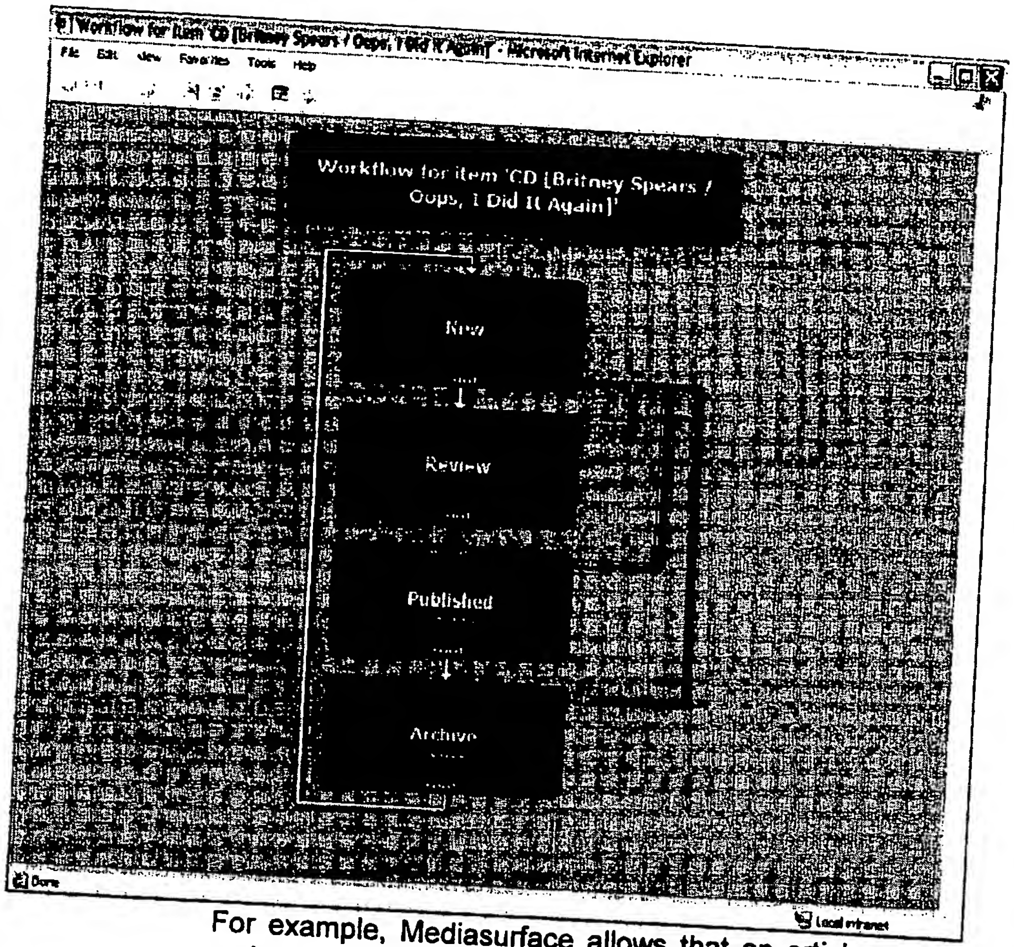
MediaSurface Interest

Centralize all data

All content — articles, news, images, font type for the display, etc — is stored in the same database. Content types are clearly defined. For example, an article is said to b long to its *Article Type* — a definition of how it is composed of fields, relations, access layers, editorial work flow, and so on. Every new piece of content is assigned to a Type, at which point it is called an Item. Items inherit their group properties from their parent Type.
The Item is then assigned to a logical part of the repository, just as files are assigned to folders in Windows.

Define the editorial flow of data entry, revision, and publication

Example of a workflow:

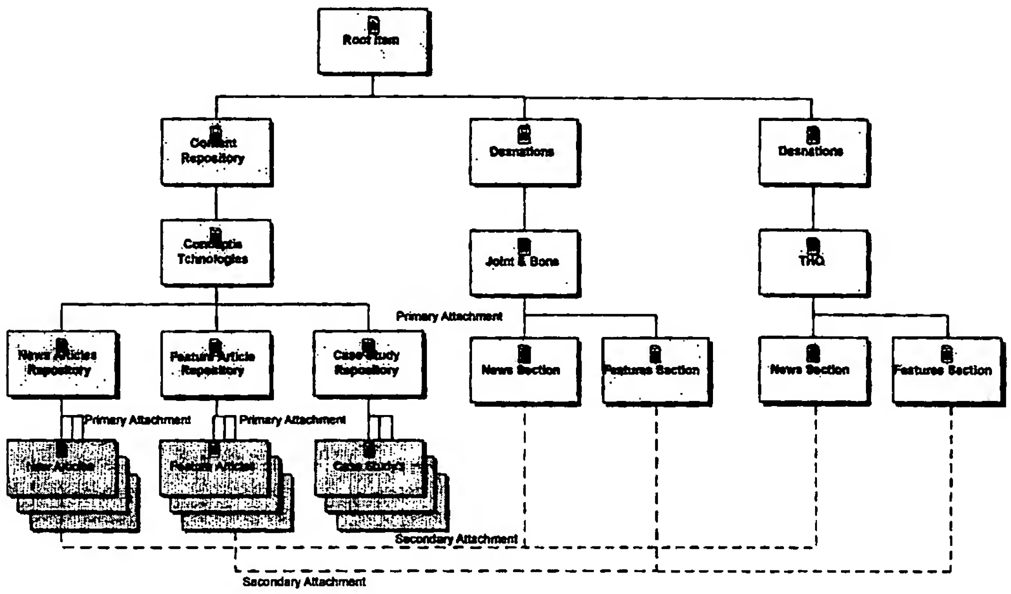

Workflow for item 'CD [Britney Spears / Oops, I Did It Again]' - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Workflow for item 'CD [Britney Spears / Oops, I Did It Again]'
New
Review
Published
Archive
Done
Local intranet

For example, Mediasurface allows that an article could be subject to review by a sub-editor before being published live on the site and then subject to review at regular intervals to ensure that its content remains current. Sign-off to the next status in a workflow can be handled

- manually
- automatically after a specified time
- at a specified time and date (one such sign-off per item)

At any stage an option can be created to allow an editor to reject the item instead of signing it off. It is then sent to another status instead, usually earlier in the workflow. When an item of a specific type reaches a specified stage in its workflow, specified groups can be informed by email. The message to be sent is configurable and can include the URL of the item which the recipient should look at.

Define and link data to a site structure



Media surface encourages the use of a single repository for all content created and entered. Each item of content may be attached at any point of any website structure as a reference and navigational aid.

Access to all content and site structure via a Java API

Through the Java API it is possible to access any item of content by navigating through the repository's branches, which are similar to folders. From a given item it is possible to view all its data and metadata, as well as its related items like images and other multimedia objects also stored in Mediasurface.

End-user browsers also gain access to the content via the Java API, but in this case only to published content and only through well-defined navigational and search paths.

Manage access right for the end user and redact r

Mediasurface offers detailed management of who should be permitted to have access to different kinds of content. Access rights can be read-only, or they may include p rmission to sign off items to their next publishing status. The sign-off permissions themselves can be specified status by status, giving a very detailed level of control over th publishing flow.

Conceptis Technol gies

History

Conceptis Technologies has developed over the past five years a number of websites using its own Content Management Systems (CMS) and delivery mechanisms (based on ColdFusion and Java Technologies). Each time a new site is to be created, the CMS and Delivery are redefined and partly redeveloped. This has been expensive in terms of resources and time. So Conceptis now has a strong interest in building a CMS that can store and manage all content and data across all its sites.

After a study of different tools, Mediasurface appeared to offer the best solution.

Conceptis Technologies' requirements

This project focuses on meeting the following requirements of Conceptis Technologies, to:

Standardize the ***editorial*** flow
Centralize all data into a single CMS
Migrate all existing data to MS
Integrate all existing and future website into MS
Use MS facilities to display content to end-users
Use MS facilities to control access rights (from both the ***editorial*** and end-user points of view)
Accelerate site development and management
Centralize maintenance
Rebuild and integrate existing services (e.g. data warehouse, spam machine)

Integration of MediaSurface into Conceptis Technologies' world

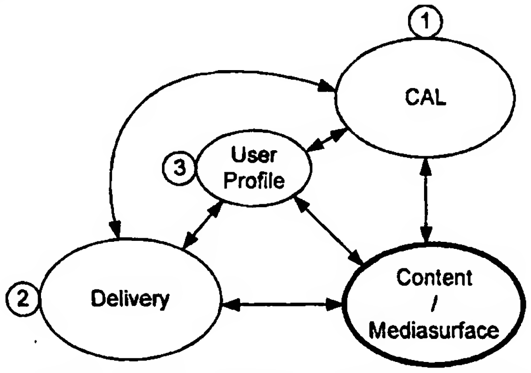
In this chapter wedefine all the different pieces that must be designed and developed in order to perform an optimal integration of MediaSurface.
In our case the central Mediasurface administrative engine is used to:

- Store and manage all the content in a single repository
- Define and manage different site structures
- Define and create model User Groups

Surrounding this we will have additional software layers to:

- Enter, edit, and manage the content repository in Mediasurface
- Deliver the content stored in Mediasurface to end-users
- Define a User management process (including authentification, personalization,storage for end-users, etc)
- Deliver peripheral services such as data warehousing and spamming

The first three of these modules and the relationship between them are shown in the following figure.



We will now define each module more specifically, starting with the Content / M diasurface Module.

Defining content

Since all the content used by Conceptis Technologies will be stored using Mediasurface, developing this module is more a process of definition. This means that our task in this case is to define our need in Mediasurface language.

Document definition

In Mediasurface the **document definition** (eg, a news article) is called **Type**. Then all **created documents** based on this Type are called **Items**.

A Type definition is composed of:

- All fields used to define this document, i.e. a news article is an agglomerate of a title, teaser, author, publication date, and so on
- A position in the main content hierarchy
- All relations with other Types. A news article can contain Images, Tables, and Paragraphs that are all Item Types on their own

These definitions are available in the Type document.

A Workflow definition is composed of:

- The different stages of publication of a document (Creation, revision, publication, archiving, and so on)
- Tasks and automatic functions triggered by each advance

These definitions are available in the Workflow document.

A Group definition is composed of:

- Access rights to documents, set by defining users and groups. A user has access to certain Types and then to certain publication stages of the associated Items and documents.
- The group's reations to Collections. A collection in Mediasurface is a complementary level of access right. An Item can be associated with a Collection when it is created. Access to a collection is restricted to certain users and groups of users just as if it were a single document..
- A set of well-defined administrative privileges.

These definitions are available in the Groups document.

Repository structure and Site structure definiti ns

In addition to d fining Typ s, Workflows, and Groups, the Content structure and Site structure must also be defined. The content structure may be thought of as a tree of folders to which Items or leaves are attached. The same concept is applicable to the site structure. When a new document (Item) is created, it is first stored in the main content structure, then given a secondary location at an appropriate place in the site structure (refer to diagram in section 2.2.3).
These definitions are available in the Content structure document and in specific Site structure documents.

Acquiring and storing content

When all of the above has been defined, we still have other tasks to do before we can enter content into Mediasurface and retrieve it, to be displayed to both company users and end-users. These tasks are the focus of the following discussion.

Entering and managing content

To enter content, Mediasurface offers three tools (***see Annex 8.2***):

- A Java client (mostly useful for administration)
- A Generic Admin Layer (GAL) designed for content users
- An Integration module for importing data that are already highly structured

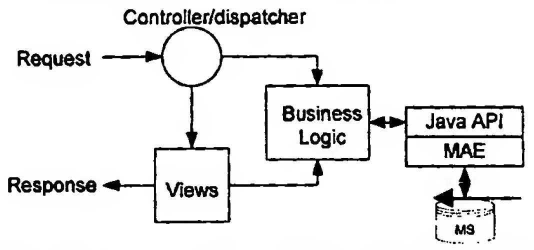
The Java client, with its heavy demand on computer resources, will be used mainly by Content managers to define Types, Groups, Stages, Structures, and so on and by Mediasurface administrators to manage sites, servers, etc.
After some testing and analysis of the Content Department requirement document (*Document CAL requirement*), it appears that the GAL provided by Mediasurface is not sufficiently useful to us, for the following reasons:

- We didn't obtain the source code, just the compiled classes
- Bugs in the GAL, e.g. user restrictions often fail
- GAL doesn't have the flexibility to add services or external modules, e.g. the WordConverter
- We have defined some non-standard document types that cannot be handled by GAL, e.g. our composite items
- Certain functional requirements defined by the content department cannot be met by GAL, e.g. our desired reporting and filtering

To be abl to enter and manage content in Mediasurfac , we will define a new module, to be called the "Conceptis T chnologies Administration Layer" (CAL), that will answer most or all of our needs. This is defined in more detail in section 4.3.

Retrieving content

To deliver content to either internal users or end-users, Mediasurface offers natively a standard approach using a controller (servlet) to dispatch requests to see certain page views (JSPs). Fulfilling these requests is likely to invoke algorithms about who sees what, called business logic, that operate in addition to the access restrictions held in the MAE. Taken altogether, the off-the-self Mediasurface system looks like this:



Although this works well if we stay in the standard Mediasurface approach, it quickly has limitations:

- One repository for each site and its content , i.e. no main repository shared by multiple sites
- Minimal or nosite personalization for users

Because we did not obtain the source code for GAL, it would be difficult to extend it to meet our needs. This compels us to develop a delivery mechanism that is:

- Modular: The same framework is available to create websites responding to a wide variety of requirements, i.e. different security layers, services, views, and so on
- Independent from Mediasurface: The same framework is available to communicate with non-Mediasurface sources of content.

The proposed Delivery module is documented in more detail below. *It is important to note that this Delivery module will be implemented twice:*

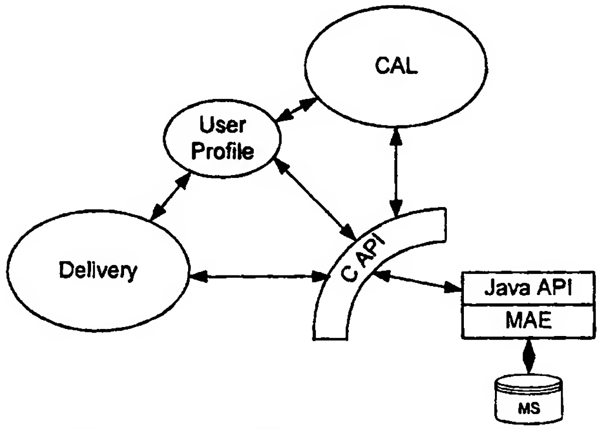
- once within the CAL to serve Conceptis users of Mediasurface (see section 4.5)
- once as the main delivery engine for all of Conceptis' other websites served to end-users (see section 4.4)

Building a Conceptis API

As we have seen, to enter or retrieve content, the standard approach is to connect to th Mediasurface Application Engine (MAE) using the Mediasurface Java API. However, for the reasons cited above, Mediasurface's own Java API is too limiting for us. We will create an interface layer between our modules and the Mediasurface Java API and MAE. This will have the following advantages:

- It clearly isolates our application modules from the Mediasurface core
- It extends or adds more flexibility and features to the Java items

Diagrammatically it will look like this::



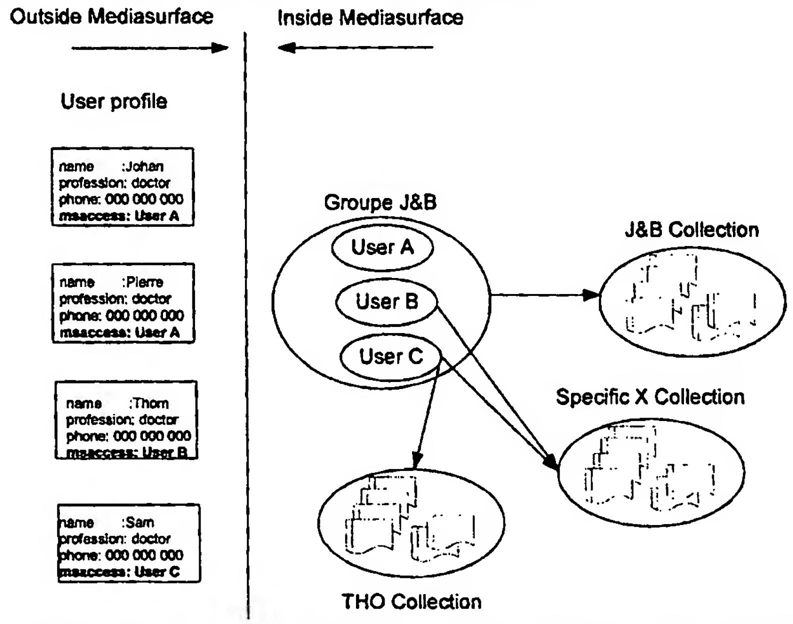
For example: If a user clicks on a link to view a news article, the delivery mechanism will request the Item. Instead of invoking the getItem() method from the Mediasurface API, it will invoke the same method but on the Conceptis Technologies API. The object coming back from this call will have the same features as a Mediasurface object plus features resulting from methods useful for the display (e.g. XML parsing logic).

Managing U rs

To access or retrieve content from Mediasurface, a User Management module is needed. Mediasurface provides both user definitions and restrictions on access rights. But personalization of websites or the creation of user profiles – these are beyond the scop of Mediasurface.
In Mediasurface, access to a given type of content is restricted to certain users or groups. When we add a new Mediasurface user, we are able to specify access rights for that user. However, we can save a lot of time by creating user *groups* and specifying access rights for those groups. In adding a new user we specify that this user is to be a member of a certain group. The new user automatically inherits all the rights of the group.

Creating profiles by mapping

A user profile in native Mediasurface is minimal: username, password, and email address.
Thus we must find a way to manage user profiles and personalization outside of Mediasurface while still using the content access controls offered by the Mediasurface software layer.
One possible solution is diagrammed below:



To determine access to Mediasurface, the external user profile will reference an existing Mediasurface user. In this example, Johan is mapped to user A from the group J&B, thereby gaining access to the J&B Collection. This is the same for Pierre. Sam is mapped to user C, who has additional rights, so Sam gains access to the J&B, Specific X, and THO Collections.
As this user management method is to be used by both the Delivery and the CAL modules, this solution must be ind pendent. That is, the User Managem nt module will run on its own server and be accessible by different services using a standard API.

D livery

The delivery engine will be in charge of managing:

- Presentation of the data
- Business logic and site personalization
- Gaining appropriate access to content
- Security layer
- Service delivery (e.g. searches, reports)

The delivery engine (framework) developed for this project will be a modular and have an open architecture. The goal is not just to deliver content from Mediasurface but also to be able to reuse this framework for any future site developed by Conceptis Technologies . Conceptis websites will be able to draw on content both from Mediasurface and from any other CMS. This is made possible by defining a standard interface for content access.

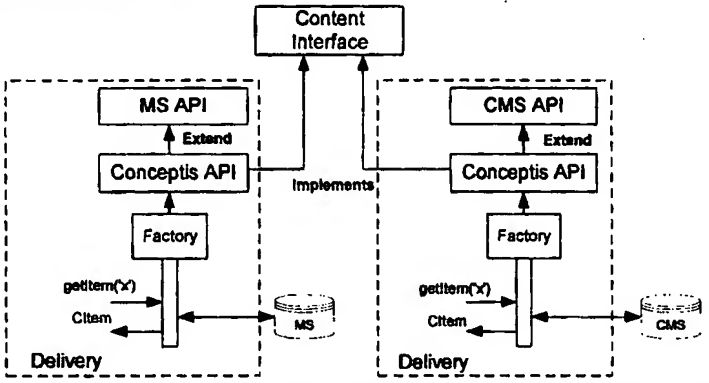


Figure 1 in the annex presents a more detailed view of the basic delivery architecture.
In this case, when the delivery business logic asks for an Item, it doesn't care about the content management system, whether it's Mediasurface or not. It always gets back a Citem defined in the content interface. So if we want to switch content management systems, we have only to change the implementation of the factory. No other changes are required on the Delivery side.

This fram work will also give us the ability to:

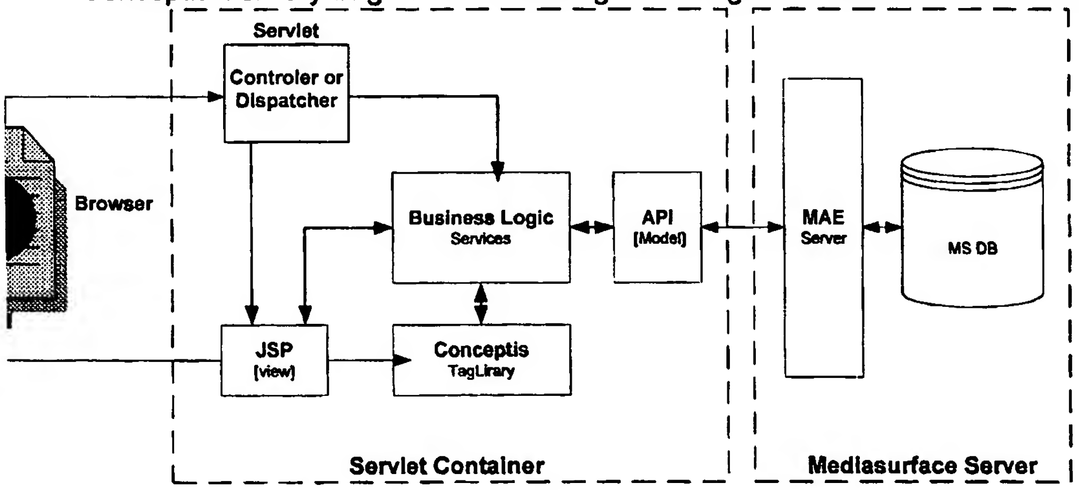
- Manage security levels
- Connect to external services (e.g. data warehouse, search)
- Define site structures and navigation from outside the repository

To make the building of the delivery engine more efficient and interesting, this module is to be defined so three teams can work independently on the tools required for website cr ation:

- Visual team (HTML integration, look and feel)
- Business logic team (planning services)
- Site team (defining site navigation, storyboard, actions)

Presenting the data

How will we display content to end-users?
We will use the combined Servlet/JSPs/Java Beans technologies — using custom tags — to access and format the presentation of content to the user. We will use a basic servlet container running several distinct processes; together they may be called the Conceptis Delivery Engine. The following schema gives a basic overview of this engine.



The request flow

Browser request

A user makes a request for content by clicking on a relevant URL. The URL refers to a content item.

Authentification by User Management

The Dispatcher verifies with the User Management module that the requesting user has the appropriate permission. The item may then be retrieved..

Applying business logic

The dispatcher passes the request to the Business Logic module, which replies with the business rules in force for displaying this kind of content this kind of user.

Dispatcher redirection to JSP

The Servlet now identifies the appropriate jJSP to forward the request to. The MS Servlet Controller achieves this by identifying the content "type" and the required "view".

Rendering content

The JSP refers to the fields of the Item. This may be done using either the Conceptis Technologies Tag Library or the API methods.

Dispatch response to browser

Once the required content has been formatted into HTML by the JSP/XSLT, the result is sent to the user's browser.

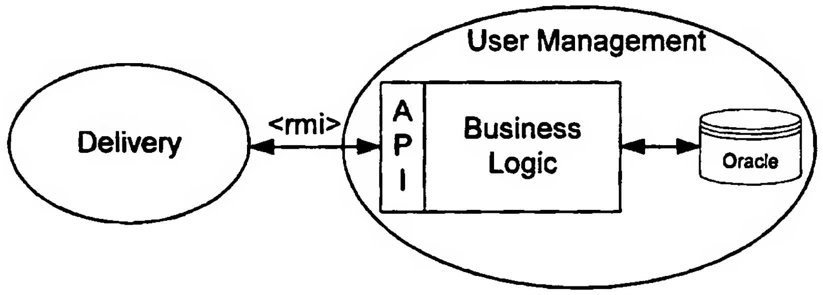
Site differentiation and personalization

Site differentiation and personalization — what content and services are displayed to different users — will be based on several intersecting datasets:

- membership in a class of users (e.g. physician registrant to theheart.org)
- personal preferences, bookmarks, subscriptions, etc
- Conceptis' business requirements (e.g. client sponsorships)
- user profile developed by Conceptis

As outlined earlier, these data are combined by a *business logic* to determine what content should be sent in response to a given browser click. Conceptis' business logic will become ever more complex as its business diversifies and matures. Therefore we will build a separate Business Logic module as part of the Delivery engine and a similar module to be part of CAL.

Information requ sted from the database of users will be processed by the rules held in the business logic component in ord r to tell Delivery what to deliv r, as shown diagrammatically here:



Making precise calls to the content repository

When the Delivery module knows what should be displayed, it can call up content from the Mediasurface repository via both the Java APIs and custom tags to be developed by Conceptis Technologies. This is the most precise way to control access rights and at the same time obtain the different Items, Items fields, and related links needed to build pages for users.

Security layer

Authentification means that a username and password are passed to the system and then verified. We have defined 3 ways to manage the security of access to the new system:

- If a website is set to use basic HTTP authentication, the Dispatcher will look to see if the user's web browser has sent an authorization HTTP header. If yes, it can extract the base 64 encrypted username/password string and authenticate the user.
- URL authentication involves a session key being appended to the URL. The session key contains within it sufficient information for the Dispatcher to be able to identify the user and authenticate him/her.
- Cookie authentication involves putting a session key into a cookie on the user's computer. If the site is set to use cookie authentication, the Dispatcher will first look to see if a session key was appended to the URL, and if not, will look for one in the user's cookie. This session key is then decrypted to extract the user's ID and other details needed to authenticate him/her.

Depending on the level of security required for a given site, we have the ability to choose any one of these 3 possibilities.

Service delivery

To handle the busin ss logic of the services available on a site, our first approach will be to use Java classes. These Java classes will be based on a Model View Controller pattern:

Input	>>	***Processing***	>>	***Output***
Get Request	>>	Process the request	>>	Display result
Controller	>>	Model	>>	View

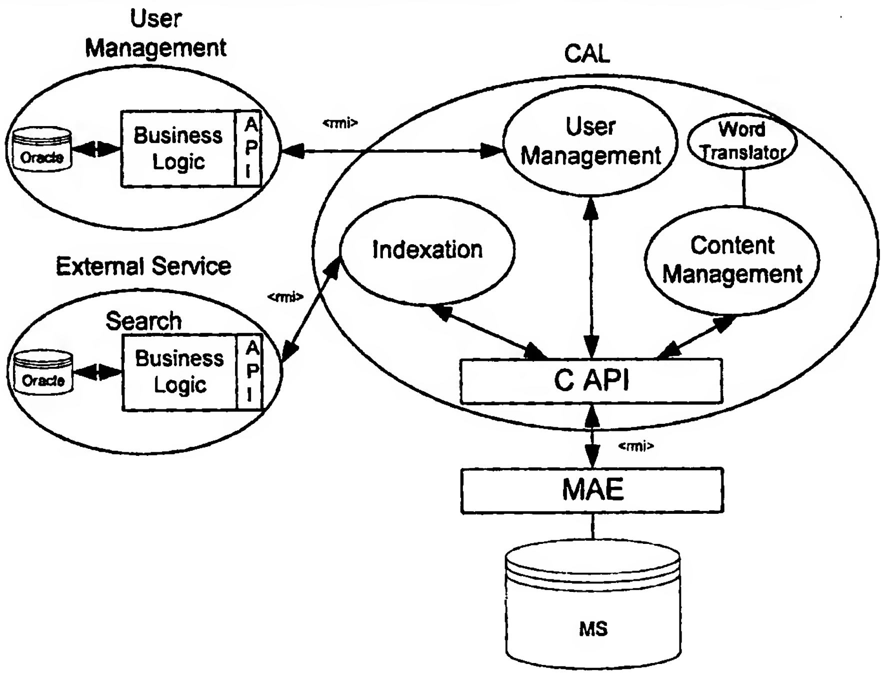
A 'controller' servlet will handle the initial request from a browser, partially process the data, and set up any required 'Model' JavaBeans to encapsulate presentation logic and state. The controller will also determine which JSP 'view' to forward the presentation of results to. The same pattern will also be used by the Java classes to make the final pres ntation of data to end-users.
This approach typically results in the cleanest separation of presentation from content, leading as well to the separation of roles and responsibilities of the developers and page designers.
A introduction of the MVC pattern is performed in the 8.3 annex.

CAL: all-in-on w bsite for M diasurfac w rkers

The Conceptis Technologies Administration Layer may be thought of as a suite of editing, managing, administering, and reporting tools combined into one "mother of all Conceptis websites". These tools, web-based, will offer different views and services to different Conceptis users. These will include:

- Content acquisition, storage, and management
- Site management
- Indexing and thesaurus management
- Personalization via business logic and user management
- Search and report interfaces
- Interfaces to data warehouse, spam machine, and project management tools
- Reference documents on today's Intranet and Extranet

Some of these functionalities are shown in the following diagram:



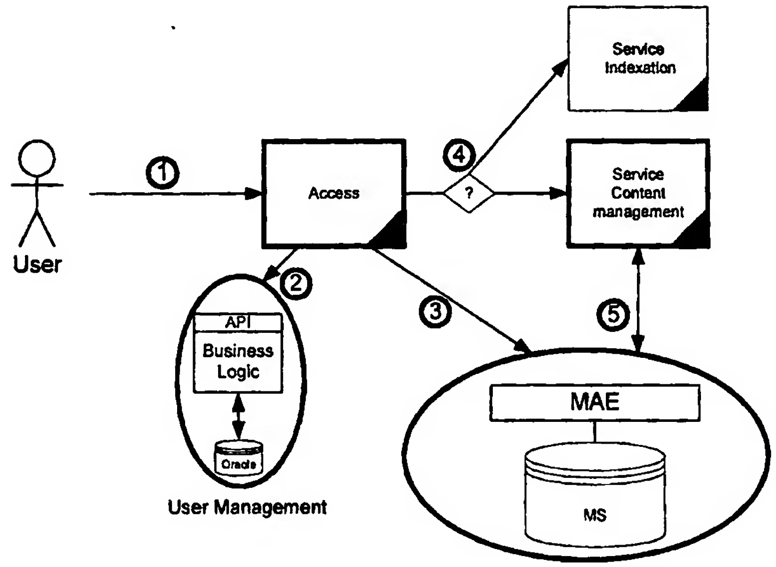
Global overview of the CAL

As the engine to be developed for the delivery is a generic site implementation process, the CAL site will employ this same engine to:

- Manage security access and personalization
- Deliver vi ws and manage actions
- Access Mediasurface's cont nt

- Use business rules for its site implem ntation.

In addition, the CAL implementation will give us the ability to reuse the xpertise gained in developing the associated website. We can attempt to define a standard Conceptis T chnologies approach to creating future websites...
Basic concept and action flow for CAL users:



1. A CAL user logs on via a logon page. The access module:

2. Validates the username and password and obtains some user personalization. In addition the user management module returns the associated model MS user and this user's associated group. The module also

3. Accesses Mediasurface to obtain this group's administration rights. The user group defines which service (e.g. content management, indexation, etc) is called upon to build the web interface. Having learned the group,

4. A specific service is initiated. This service uses the user administration rights already defined in Mediasurface to define the range of actions available for this CAL user.

5. The user is then able to work in a customized environment. For example, specific filters can be defined for displaying a list of Items and then stored in the user management database.

6. This method allows different services to have different URLs and different welcome pages. For example, an editor would bookmark the URL for the content manag ment suite of functions. An administrator would bookmark this URL and also mark the URL for the administrative suite of functions.

Content management

Content management functions available to internal users

- Acquire, create, and update content
- Follow specific workflows and schedules
- Index content
- Manage websites and website content
- Navigate through the repository either by Type or by Site and site assignment
- Run searches, obtain reports on content holdings
- Preview a variety of output formats (e.g. without images, PDA, print, etc)
- Make rich use of relations and links
- Standardize storage and views for each new Type as it is defined
- Hide and/or remove unwanted content

The conversion from Word to Mediasurface and back again

The creation of text items is performed using the CAL web-based interface or Microsoft Word. For some document types (e.g. articles), the creation, viewing, and updating is carried out with Word. A Word-based macro will be developed that will allow users to edit and update documents. It will:

- Define tags corresponding to Mediasurface fields
- Save Word documents directly as Mediasurface items
- Allow users to obtain a copy of a text-based item in its native Word format
- Allow users to modify the Word document, re-save, and modify the corresponding Mediasurface item

Handling media items

- CAL interface is used to enter, manage, and store all document types, including lists, forum messages, and multimedia items

- Mediasurface does not directly store all large binary objects, but it stores all metadata and allows thes items to b managed like any other
- Mediasurface allows media items to be viewed both as components of complex publishing projects and as stand-alone items in a media library

Indexing

Indexing will be managed partly within Mediasurface and partly outside, because:

- Indexing may be performed at any one of several publication stages, i.e. not in a generic Mediasurface way
- Mediasurface does not support the use of thesaurus lists
- Mediasurface does not support custom searches like Cardiosearch

However, *all* aspects of the indexing service will be accessed via CAL:

- Web interfaces to assign and revise keywords for all content
- Web interface to manage thesaurus lists
- Web interface to test and manage custom searches

Managing parallel workflows

Unfortunately Mediasurface does not permit us to define parallel workflows, not does it have build-in scheduling functions. So Conceptis will devise several work-arounds to meet our requirements. Parallel workflows include those for

- indexing
- securing copyright permissions
- transcribing
- summarizing
- translating

Items will have internal fields as flags to define the different stages of the parallel workflow, so that these stages b come visibl to users, e.g. indexed or translated. Access to the parall l workflows and the setting of such flags will be carried out via CAL.

Services run from outside the MAE

CAL will also be the user interface for managing services built and deliv r d external to the main Mediasurface MAE, including but not limited to th saurus management, data warehouse, custom searching (e.g. Cardiosearch), spamming, project management tools, and so on.

User management

This service will give administrators the ability to manage, via a CAL interface, both site users and internal users. (Note that information about individual users is stored in the independent database maintained by the user management module, whereas the access rights of model users are stored in Mediasurface and managed via its Java client software.)
Th CAL interface will:

- Allow user Information to be entered and managed
- Enforce the concordance between Mediasurface's model users and model groups and the real users held in the external user database

All of the above requirements and definitions found in section 4.5 are defined in the CAL requirements document.

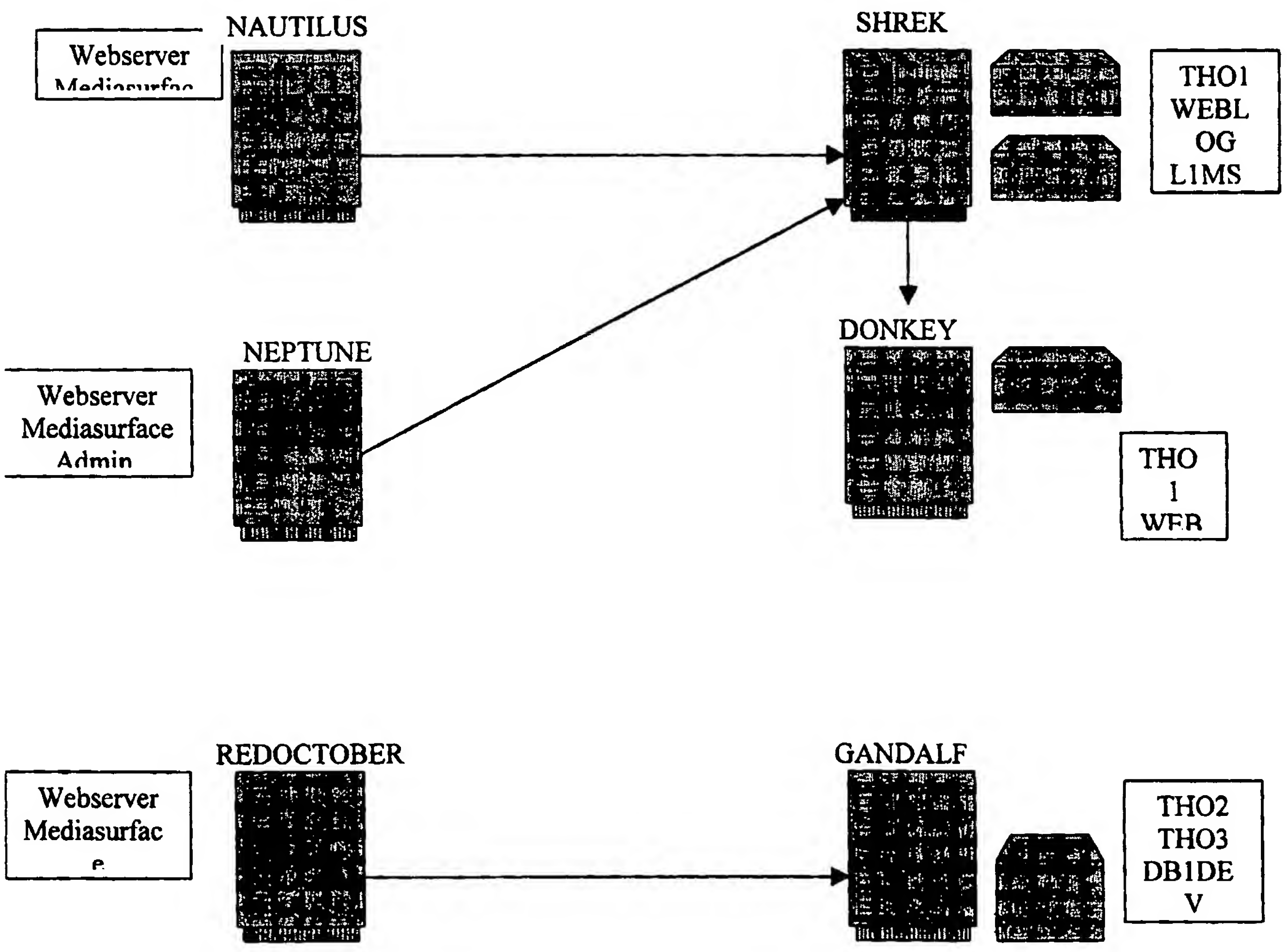
Surrounding Module

- Search
- Data Warehouse
- Banner management
- User management

Hard Ware

We will now define the desir d and defined hard ware implem ntation. In a first step we will display a view of the implementation architecture. This is composed from a set for the d velopment (RedOctober, Gandalf) and production (Nautilus, Shrek, Neptun, Donkey). The details of this computers and their utilization is then details in the next section (Computer in details).

Hard ware map



Computer in details

E3500 –Live & Application Server 'Nautilus'

This server is to be used as a platform for both the Mediasurface software and the application software. It will take the role of a Mediasurface live server, serving pages to the web.
Th box specifications are:

2 x 400 MHz processors
2.5 GB RAM
2 x 9 GB HD configured as a 9GB mirrored pair
2 x 36 GB HD configured as a 36 GB mirrored pair

It is likely that this server will handle the role assigned to it without difficulty.

E1 – Mediasurface Staging & Application Server 'Neptune'

This server is to be used as a platform for both the Mediasurface software and the application software. It will take the role of a Mediasurface staging server, serving the **admin** layer pages and allowing use of the java client.
The box specifications are:

1 x 166 MHz processor
750 MB RAM
5 x 9 GB HD in a separate disk pack ***(to be confirmed)***

It is likely that this server will cope with the Mediasurface processes, but it is also likely that it will be a performance bottleneck when the java application server needed to serve the pages is taken into consideration. **This box is below the Mediasurface minimum specification, but due to financial considerations and the fact that external users will not be effected, it is has been decided to test its suitability.**

E280R – Production Databas Server – 'Shrek'

This server is Conceptis n w production database server that is used as the database s rver for the current Conceptis sites.
The box specifications are:
1 x 750 MHz processor
2 GB RAM
2 x D1000 disk array set up as a mirrored pair
14 x 9 GB HD with 7 drives in each disk pack
As previously stated this database server will have to run two versions of oracle (O8.1.7.2.0 and O9.0.1.2.0).
This will allow Conceptis to alter the availability of system resources to the staging and live environments so that a process occurring within staging will not impact on live. It is thought that the server will cope with these extra requirements, as the present usage is understood to make little impact on system resources.

E250 – Fail Over Database Server 'Donkey'

This server is to be used in the event of a failure of the production box, so it will have the same loads both in terms of data and usage should a fail over situation ever occur.
The box specification is:
1 x 400 MHz processor
1.5 GB RAM
1 x D1000
6 x 9 GB HD (a plus)
This server is considerably less powerful than the production E280R that it would replace, so it should not be expected to provide the same level of performance. The memory usage of several oracle instances must be looked at to ensure that one giga byte is enough.

E3500 – Mediasurfac & Applicati n Server 'RedOctober'

This server is to be used as a platform for both the Mediasurface software and the application software. It will take the role of a Mediasurface Dev and Test server.
The box specifications are:

1 x 400 MHz processors
1 GB RAM

This server will be used to run four or more instances of Mediasurface, and all the application server processes to support these as well. There are no performance expectations for this box, so it should perform as required.

E280R – Development Database Server 'Gandalf'

The Development database server is required to have one instance and four schema for each of the four Development Mediasurface schema. It will also need to run the test environment for the present sites.
The box specification is:

1 x 750 MHz processor
1.5 GB RAM
1 x 36 GB HD
1 x storage multi pack with 3 x 36 GB HD 3x9 GB HD

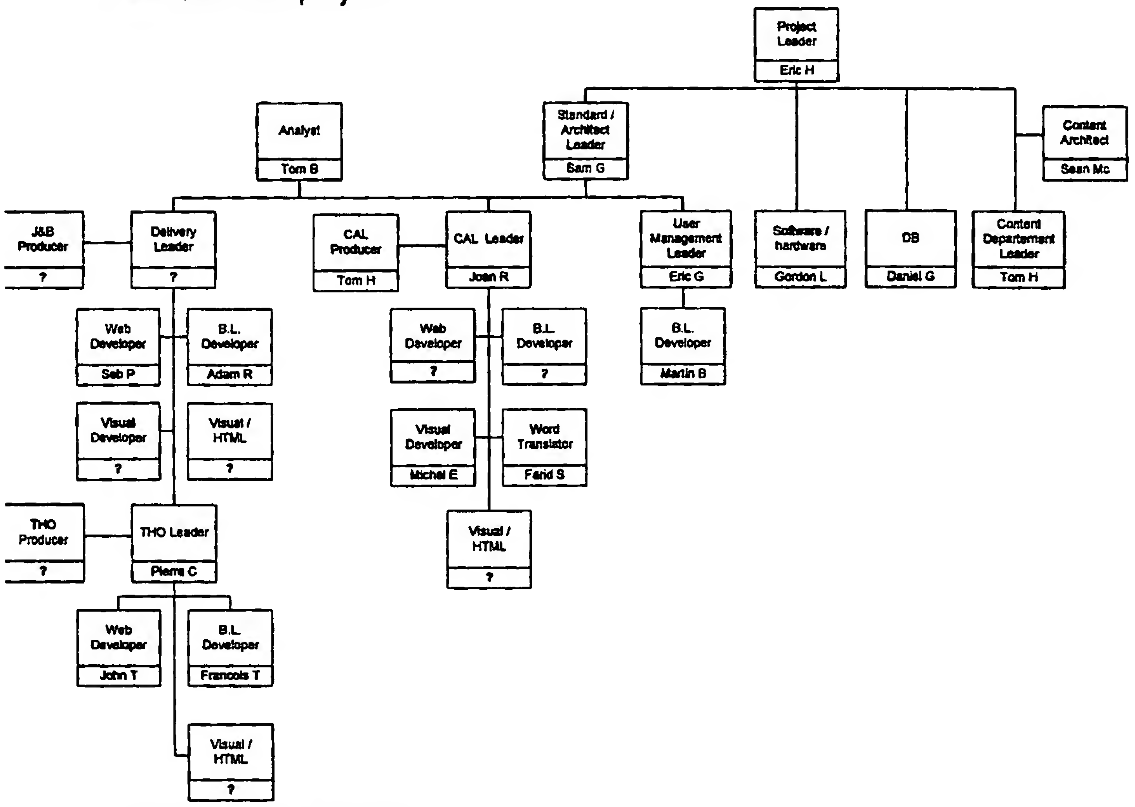
This database sever has almost the same specification as the production server, but is unlikely to have the same load placed on it, so it is expected to perform well. Depending on how large the database grows, disk space may become an issue because there will be two copies of it, one for each instance.

We must perform a migration process to get this hardware configuration. This is describe in the Daniel's document named *Media Surface Hardware.doc* taking place under *\\Files1-mtl\Programmeurs\MediaSurface\Architecture\Design*

Task definition and responsibility

Res urces diagram

This diagram shows us the basic technical resources and there responsibility for the Mediasurface project.



Development Approach:

- Delivery

  The first step after the framework and architecture validation is to develop the generic tools.

  - Conceptis API (to isolate Medisasurface from the solution define)
  - Basic delivery components (controller, dispatcher…)
  - Basic services

  In fact this will give us the ability to get a first version of all module communicate together.
  After that the first site development can start. For each site this will follow the same pattern:

  - 3 developers (one for the web, one for the Business Logic and one for the visual)
  - 1 integrator (HTML and look and fe l)

  - 1 producer (site definition, navigation ...)

  The tools and code produce by the three developers is not site specific but can in fact be reused ***throwsover*** the future site.

- CAL

  The CAL will use the generic delivery development and resource pattern. Then depending of the module priority (Content, indexation, user management,...) the development will take place in phases.

- External services

  Switching the resources availability the development is done in groups of at least two resources.

- Migration

  Migration is a one shot process. One resources of the development team and two of the content department for at least two weak will be necessary to do this job.

- Content

Dani I Garant:
Installation and configuration of the MediaSurface DB

- BD manager

Gordon Lamb
Installation and configuration of the MediaSurface Server

- System administrator

Thomas Bennett:
General analyze and design

- Delivery and services analyst

Marc Belanger:
Content Architecture (Document structure, site structure, storage, data migration)

- Content model definition

Eric Hechinger:
Application architecture (CMS core, web server, Java Web container)

- Project manager

Sam Guembour:

- General analyze and design architect.
- Team lead on the delivery side

Adam Ramadan:

- Back end and business logic Java developer

Sebastien Proulx:

- Front end and web Java developer

Farid S

- Translator (Microsoft Word to Mediasurface)
- Migration process

Michel E

- Visual developer

Joan Roch:

- Team lead for the CAL
- Java developer

Thomas Holzinger:

- Content definition lead
- CAL producer

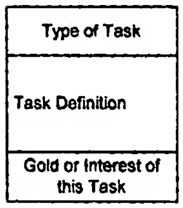
Action Plan

The following section will introduce the action plan for the Content and Delivery side. A time line will then be extract from this action plan.

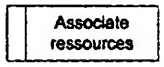
Action

This action plan is a graphical representation.
Legend:

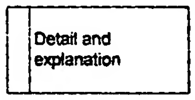
Type and number of resources involved (Blue is for the development team, and green is for the content department team)



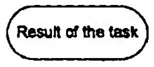
Task representation



Prerequisite Task or Task result

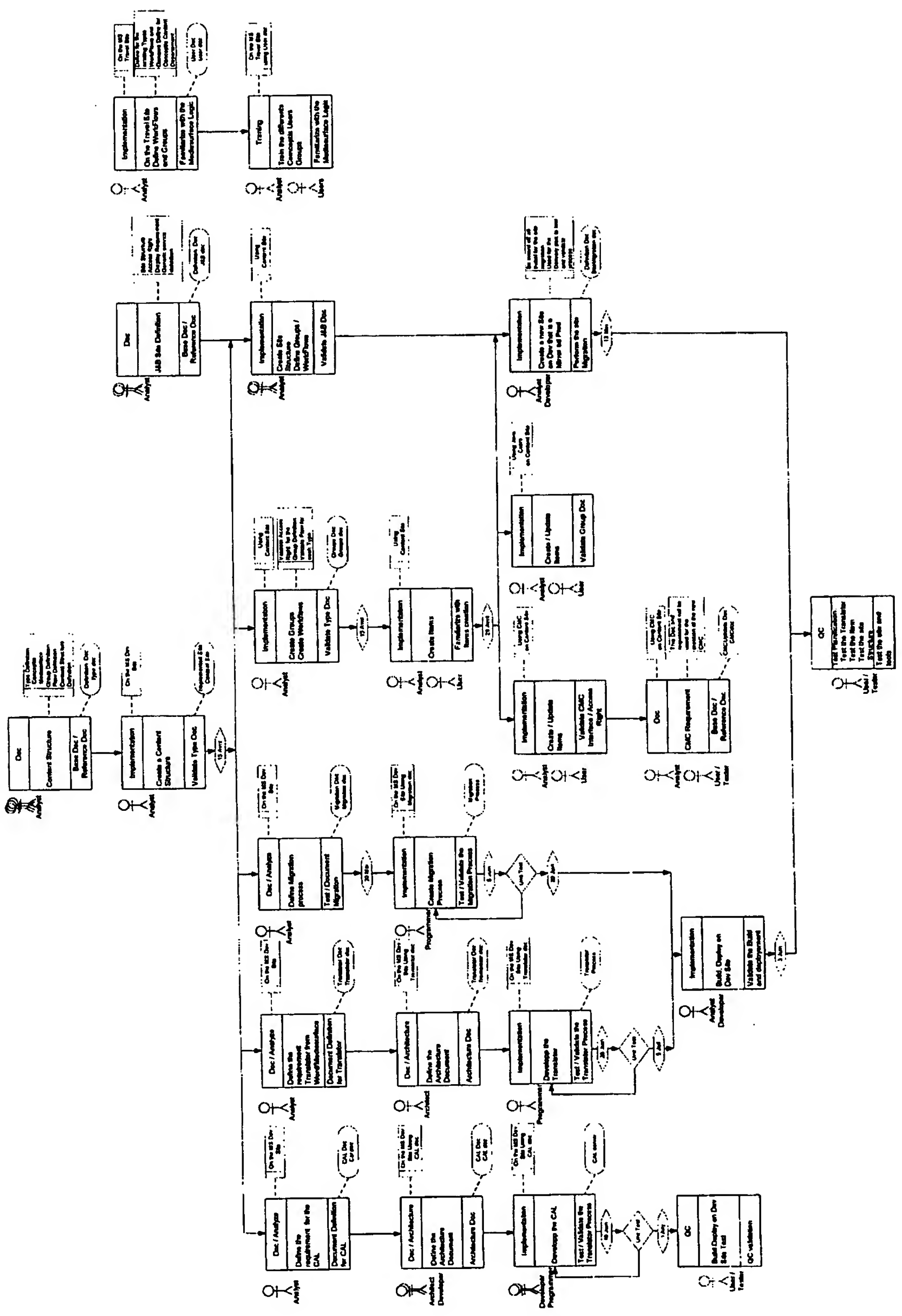


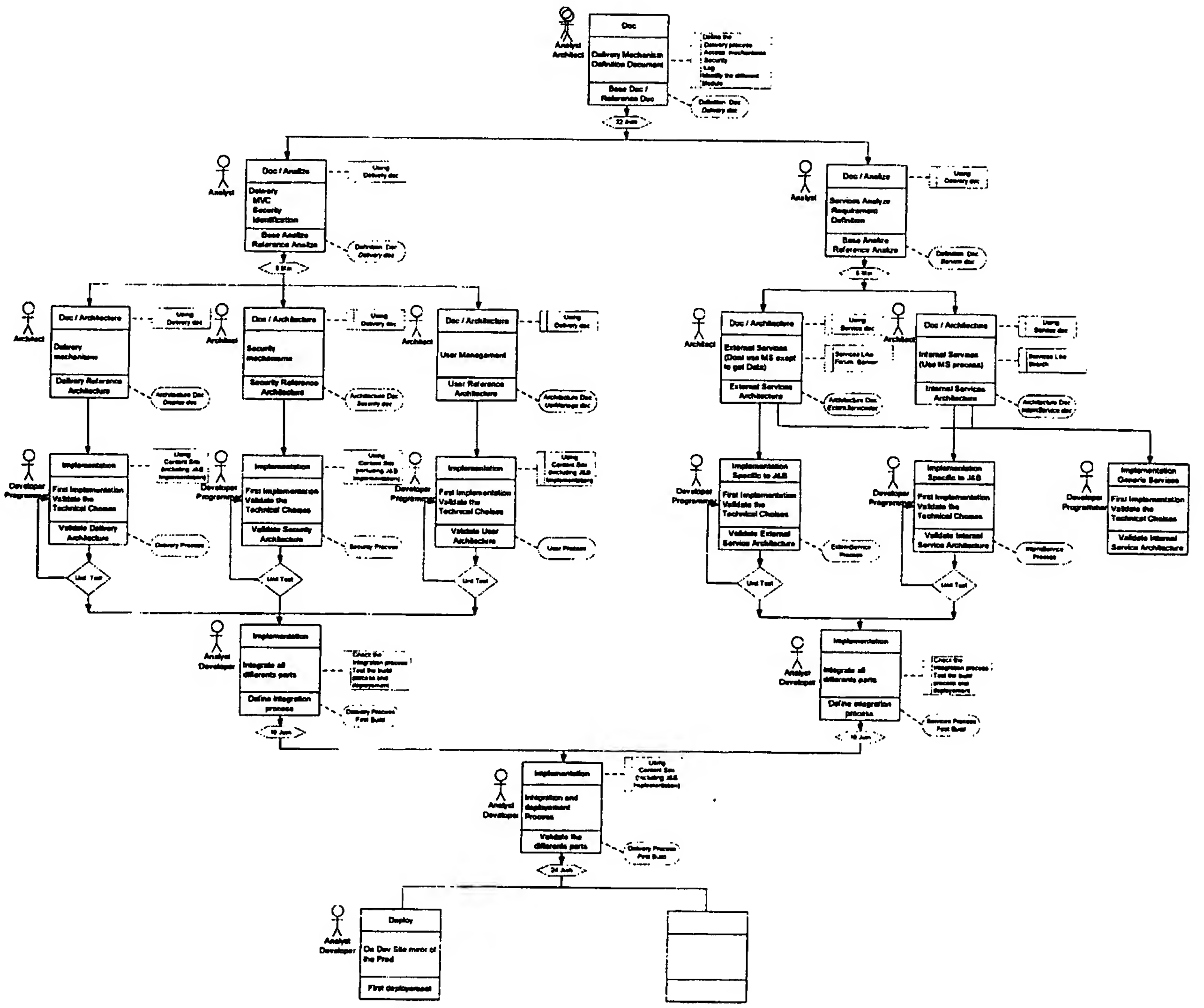
Task detail or explanation



Task result

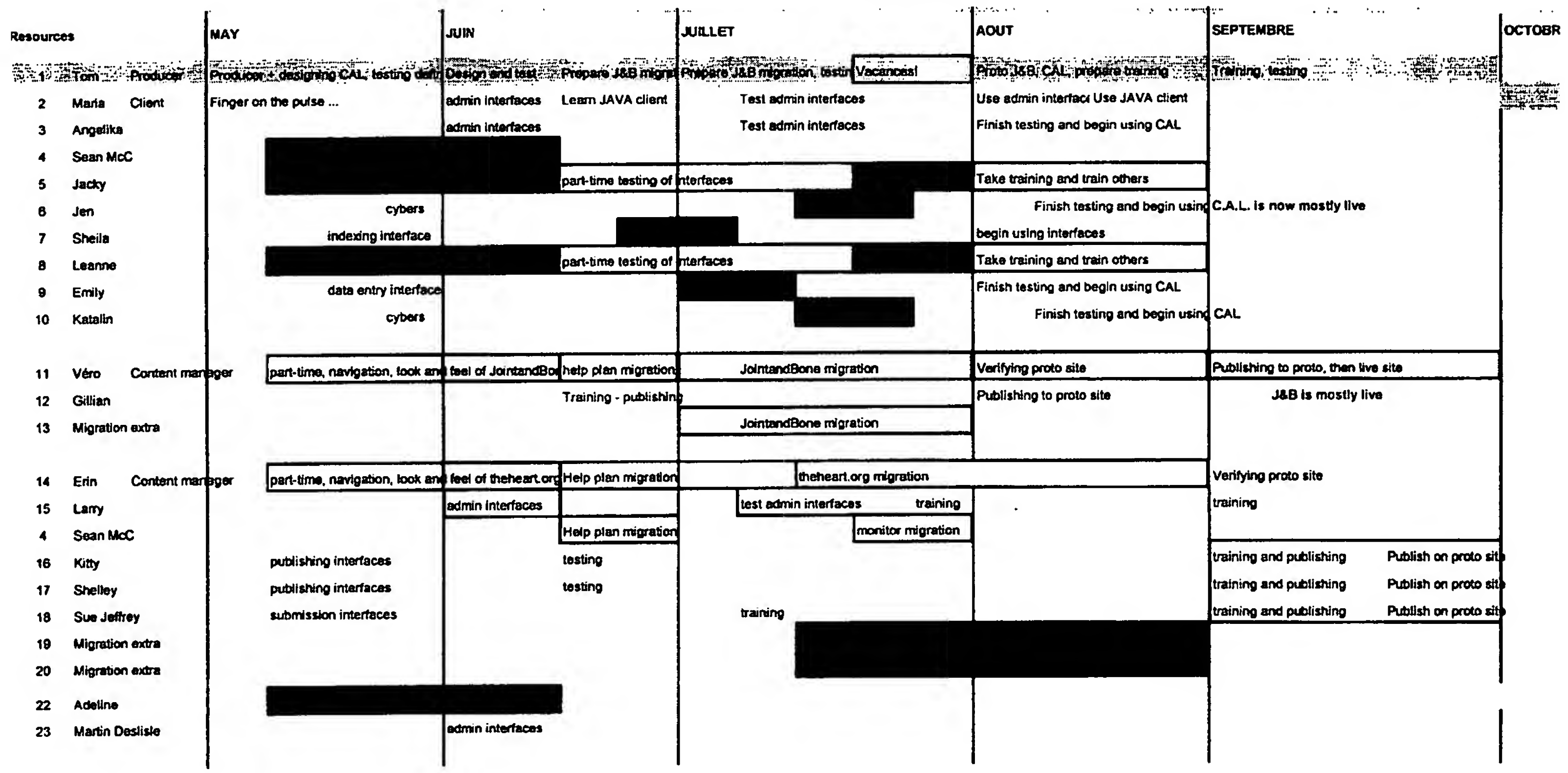
The first diagram is for the content and the second for the delivery.




Analyst Architect
Doc
Delivery Mechanism Definition Document
Base Doc / Reference Doc
Definition Doc Delivery doc
22 Juin
Analyst
Doc / Analize
Delivery MVC Security Identification
Base Analize Reference Analize
Using Delivery doc
Definition Doc Delivery doc
Doc / Analize
Services Analyze Requirement Definition
Base Analize Reference Analize
Using Delivery doc
Definition Doc Service doc
Architect
Doc / Architecture
Delivery mechanisms
Delivery Reference Architecture
Security mechanisms
Security Reference Architecture
User Management
User Reference Architecture
External Services (Dont use MS except to get Data)
External Services Architecture
Internal Services (Use MS process)
Internal Services Architecture
Developer Programmer
Implementation
First Implementation Validate the Technical Choises
Validate Delivery Architecture
Validate Security Architecture
Validate User Architecture
Delivery Process
Security Process
User Process
Implementation Specific to J&B
Validate External Service Architecture
Validate Internal Service Architecture
Implementation Generic Services
Unit Test
Analyst Developer
Integrate all differents parts
Define integration process
Delivery Process First Build
Services Process First Build
10 Juin
Integration and deployment Process
Validate the differents parts
Delivery Process First Build
24 Juin
Deploy
On Dev Site mirror of the Prod
First deployement

Time

Resources						MAY 6-May	13-May	20-May	27-May	JUIN 3-Jun	10-Jun	17-Jun	24-Jun	JUILLET 1-Jul	8-Jul	15-Jul	22-Jul	29-Jul	AOUT 5-Aug	12-Aug	19-Aug	26-Aug	SEPTEMBRE 2-Sep	9-Sep	16-Sep	23-Sep
	1	Joan R				Design	Design	Design	Design	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl
	2	Eric H / ?						Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl
	3	Farid S	Translator						Design	Design	Impl	Impl	Impl													
	4	Marc B				Analysis	Analysis	Analysis	Impl	Impl	Impl	Impl	Impl													
	5	Marc B		C struct	C struct																					
General Development	6																									
	7	Sam G	Sam G / ?			Design	Design	Impl	Impl	Impl	Impl															
	8	Thom B	Adam R			Design	Design	Impl	Impl	Impl	Impl															
	9		Sebastien P				Impl	Impl	Impl	Impl	Impl															
	10	Eric G						Analysis	Analysis	Analysis	Impl	Impl	Impl	Impl												
	11	Marc B							C struct	C struct	C struct	C Migrati	C Migrati	C Migrati	C Migrati											
J&B	12	Farid S																								
	13	Sam G/?										Impl G	Impl G	Impl G	Impl G	Impl G	Impl G	Impl G								
	14	Adam R										Impl Sp	Impl Sp	Impl Sp	Impl Sp	Impl Sp	Impl Sp	Impl Sp								
	15	Sebastien P										Impl Sp	Impl Sp	Impl Sp	Impl Sp	Impl Sp	Impl Sp	Impl Sp								
	16	Marc B								C/Site stru	C/Site stru	C Migrat	C Migrat	C Migrat	C Migrat	C Migrat	C Migrat	C Migrat	C Manag	C Manag	C Manag	C Manag				
THO	17	Farid S											C Index	C Index	C Index	C Index	C Manag	C Manag								
	18	Pierre C										Serv Arch	Serv Arch	Design	Design	Impl	Impl	Impl	Impl	Impl	Impl	Impl				
	19	John T										Serv Arch	Serv Arch	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl				
	20	Francois T										Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl	Impl					
CTC / KC	21															C/Site stru	C/Site stru	C Migra	C Manag	C Manag	C Manag	C Manag				
360	22																		C Migra	C Manag						
	23													Del Archi	Del Archi	Serv Arch	Serv Arch	Design	Impl	Impl	Impl	Impl				
	24															Impl	Impl	Impl	Impl	Impl	Impl					
				3	2	4	5	7	9	10	10	13	14	14	13	13	13	13	10	10	9	7				


Resources
MAY
JUIN
JUILLET
AOUT
SEPTEMBRE
OCTOBR
1 Tom Producer
Producer - designing CAL, testing
Design and test
Prepare J&B migra
Prepare J&B migration, testin
Vacances!
Proto J&B CAL, prepare training
Training, testing
2 Maria Client
Finger on the pulse ...
admin interfaces
Learn JAVA client
Test admin interfaces
Use admin interfac Use JAVA client
3 Angelika
admin interfaces
Test admin interfaces
Finish testing and begin using CAL
4 Sean McC
5 Jacky
part-time testing of interfaces
Take training and train others
6 Jen
cybers
Finish testing and begin usin
C.A.L. is now mostly live
7 Sheila
indexing interface
begin using interfaces
8 Leanne
part-time testing of interfaces
Take training and train others
9 Emily
data entry interface
Finish testing and begin using CAL
10 Katalin
cybers
Finish testing and begin usin CAL
11 Véro Content manager
part-time, navigation, look and feel of JointandBo
help plan migration
JointandBone migration
Verifying proto site
Publishing to proto, then live site
12 Gillian
Training - publishing
Publishing to proto site
J&B is mostly live
13 Migration extra
JointandBone migration
14 Erin Content manager
part-time, navigation, look and feel of theheart.org
Help plan migration
theheart.org migration
Verifying proto site
15 Larry
admin interfaces
test admin interfaces
training
training
4 Sean McC
Help plan migration
monitor migration
16 Kitty
publishing interfaces
testing
training and publishing
Publish on proto sit
17 Shelley
publishing interfaces
testing
training and publishing
Publish on proto sit
18 Sue Jeffrey
submission interfaces
training
training and publishing
Publish on proto sit
19 Migration extra
20 Migration extra
22 Adeline
23 Martin Deslisle
admin interfaces

Time Line / Resources

<u>Resources estimate for the content and development team.</u>

<u>Time Line</u>

List of primary task:

- Preliminary analyse
- Set up
- Requirement analyse
- MS Specific development
- ConceptisTechnologies site specific development
- J&B site

Preliminary analyse

This was done during the validation of the Mediasurface product.

Set up

This phase has for focus to:

- Set-Up the software and hardware.
- Define a development environment
- Define the production and development environment including the server migration.

Requirement analysis

Th goal of this phase is to move forward on all the different aspects of the implementation process of Mediasurface. This will also give us the ability **to validate the primary analysis of the MS** consultant team so that we can begin the implementation process of J&B and the architecture and development of the surrounding processes.

Task Name	Duration	Start	Finish	Predecessors	Resource Names	Deadline
Preliminary Analysis	6 days?	Mon 3/18/02	Mon 3/25/02			NA
Requirement Analysis	33 days?	Wed 2/13/02	Mon 4/1/02			NA
Mediasurface content Objects	28 days?	Wed 2/13/02	Fri 3/22/02			Fri 3/22/02
Types	23 days?	Wed 2/13/02	Fri 3/15/02			Fri 3/15/02
Conceptis Type definition	21 days?	Wed 2/13/02	Wed 3/13/02			Wed 3/13/02
Doc Type	19 days?	Wed 2/13/02	Mon 3/11/02			NA
Definition	17 days?	Wed 2/13/02	Thu 3/7/02		marc b	NA
Validation	2 days	Fri 3/8/02	Mon 3/11/02	13	marc b[50%]	NA
Item Type	18 days	Wed 2/20/02	Wed 3/13/02			NA
Definition	8 days	Wed 2/20/02	Fri 3/1/02		marc b	NA
Validation	2 days	Tue 3/12/02	Wed 3/13/02	14	marc b[50%]	NA
MS Type definition	11 days	Fri 3/1/02	Fri 3/15/02			Fri 3/15/02
Site type	8 days	Mon 3/4/02	Wed 3/13/02			NA
Definition	5 days	Mon 3/4/02	Fri 3/8/02		marc b	NA
Validation	1 day	Wed 3/13/02	Wed 3/13/02	20	marc b	NA
Repository types	11 days	Fri 3/1/02	Fri 3/15/02			NA
Definition	8 days	Fri 3/1/02	Tue 3/12/02		marc b[50%]	NA
Validation	2 days	Thu 3/14/02	Fri 3/15/02	23,21	marc b	NA
Repository	7 days	Thu 3/14/02	Fri 3/22/02			Fri 3/22/02
Content Repository	7 days	Thu 3/14/02	Fri 3/22/02			Fri 3/22/02
Structure	3 days	Thu 3/14/02	Mon 3/18/02		marc b	NA
Relation	2 days	Tue 3/19/02	Wed 3/20/02	27	marc b	NA
Validation	2 days	Thu 3/21/02	Fri 3/22/02	28	marc b	NA
Workflow	20 days	Thu 2/14/02	Wed 3/13/02			Tue 3/19/02
Analyse	12 days	Thu 2/14/02	Fri 3/1/02		tom h	NA
Validation	3 days	Mon 3/11/02	Wed 3/13/02		tom h	NA
Groups	8 days	Wed 3/6/02	Fri 3/15/02			Tue 3/19/02
Analyse	3 days	Wed 3/6/02	Fri 3/8/02	31	tom h	NA
Validation	2 days	Thu 3/14/02	Fri 3/15/02	32	tom h	NA
Content	13 days?	Wed 3/6/02	Fri 3/22/02			Fri 3/22/02
MS Word processor	1 day?	Wed 3/6/02	Wed 3/6/02		Tom B	NA
Web based interfaces for Content input	5 days?	Mon 3/18/02	Fri 3/22/02		tom h	NA
Personalisation	5 days?	Wed 3/6/02	Tue 3/12/02			Fri 3/22/02
User registration	2 days	Mon 3/11/02	Tue 3/12/02		eric h	NA
Security	1 day?	Wed 3/6/02	Wed 3/6/02		Tom B	NA
Site access	1 day?	Wed 3/6/02	Wed 3/6/02		Tom B	NA
Services	8 days	Wed 3/13/02	Fri 3/22/02			Fri 3/22/02
Personalization	2 days	Wed 3/13/02	Thu 3/14/02		eric h	NA
Search	2 days	Fri 3/15/02	Mon 3/18/02	44	eric h	NA
Case study	1 day	Tue 3/19/02	Tue 3/19/02	45	eric h	NA
Collection	1 day	Wed 3/20/02	Wed 3/20/02	46	eric h	NA
Banner	1 day	Thu 3/21/02	Thu 3/21/02	47	eric h	NA
Forum	1 day	Fri 3/22/02	Fri 3/22/02	48	eric h	NA
Validation by Mediasurface Team	6 days	Mon 3/25/02	Mon 4/1/02			NA
Types	0.5 days	Mon 3/25/02	Mon 3/25/02		Mediasurface	NA
Repository	0.5 days	Mon 3/25/02	Mon 3/25/02	51	Mediasurface	NA
Workflow	0.5 days	Tue 3/26/02	Tue 3/26/02	52	Mediasurface	NA
Groups	0.5 days	Tue 3/26/02	Tue 3/26/02	53	Mediasurface	NA
Migration	0.5 days	Wed 3/27/02	Wed 3/27/02	54	Mediasurface	NA
Content	0.5 days	Wed 3/27/02	Wed 3/27/02	55	Mediasurface	NA
Final revision	0 days	Mon 4/1/02	Mon 4/1/02	56	Conceptis	Mon 4/1/02
Mediasurface specific development	17 days?	Wed 3/6/02	Thu 3/28/02			NA
Conceptis sites specific development	1 day?	Wed 3/6/02	Wed 3/6/02			NA
Setup	17 days?	Mon 2/18/02	Tue 3/12/02			NA

The tasks can be divided into content and delivery. These two sections will now be developed.

Content

All ConceptisTechnologies' document types, staging of editorial processes, and groups of people involved in the publishing flow will be d fined in a document. This docum nt will describe Types, Workflows, and Groups in a Mediasurface-standard approach. This will be our reference document during the implementation process.
A repository content based on the defined types will be defined. This repository will contain all the primary attach points for all the content entered into Mediasurface.
All external process used to get or display content for the content department will also be covered here.

Delivery

The delivery part of the development includes personalization and services.
Personalization
A standard document for registration will be defined. This document will define the standard procedure for the registration flow and the way that user data are stored and related.
A document defining the different ways for performing security and site access using Mediasurface will also be defined.
Services
A first service analyse will be performed.

Objective of this phase:
Address the surrounding tasks for integrating Mediasurface into the Conceptis Technologies world.
Familiarize ourselves with the software and the logical approach used by Mediasurface. This will be performed during the validation phases.

When this phase and its objectives have been accomplished, we will be abl to mov forward to the n xt phases in good shap :

- Begin the architecture and development of the surrounding processes (MS-specific development and Conceptis Technologies site-specific development).
- Begin the implementation process of J&B.

MS Specific development

ConceptisTechnologies site specific development

J&B site

Annex

Delivery Architecture

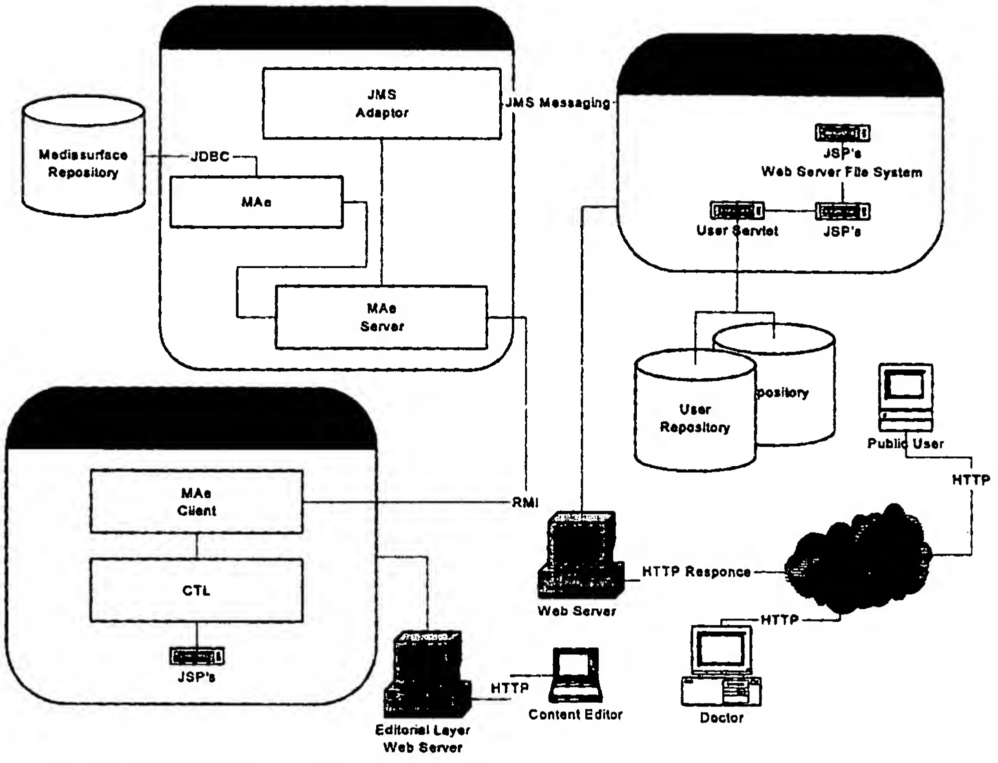


Figure 1

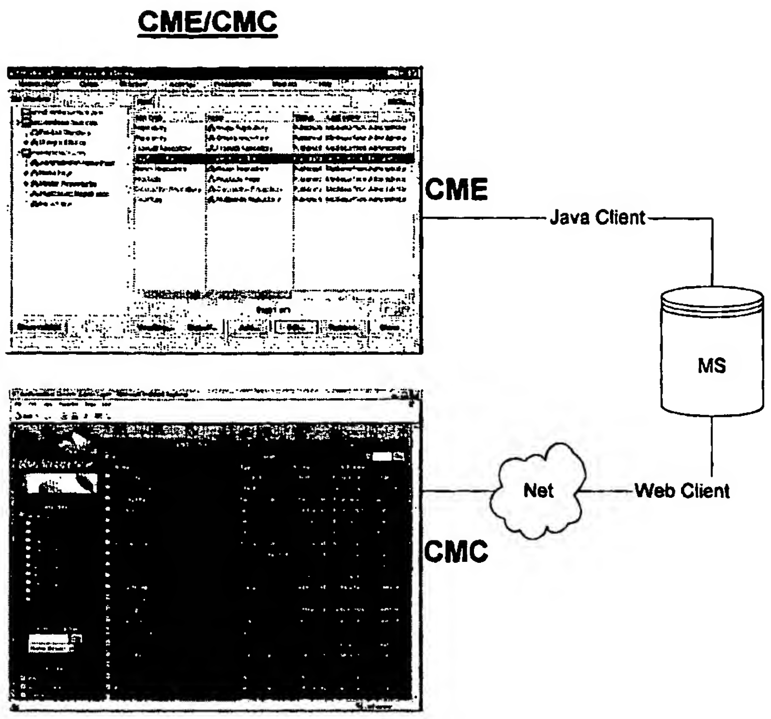


Figure 2

Content Management Engine (CME)

The Content Management Engine is the core of Mediasurface and is comprised of three main elements.

Rules Console

- Define the content types on a site. Each type has its own structure of fields, verification rules, and workflow.
- Define the statuses that make up a workflow and the collections to which items belong.
- Control user access to specific content types and workflow stages.
- Control user access to specific parts of the Rules Console.
- Define groups of users, with access rights based on roles. For example, a user could be a contributor or section editor.
- Create templates to convert item information into the required delivery format.
- Define commonly used presentation components. Such components only need to be changed once to aff ct all templates

that us them.

•

Synchronization Server
The Synchronization Server controls client access to Mediasurface, synchronizes objects between the clients and servers, and handles the locking of objects for editing.

Task Servers
The Task Servers implement all server-side operations that are carried out at timed intervals, as well as removing the burden of some background processing from the Rules Console and the APIs. These operations include the following:

- Signing off items automatically after a specified interval or at a specified time.
- Alerting groups of users by e-mail when items are signed off.
- Relocating items when the location to which they were attached is deleted.

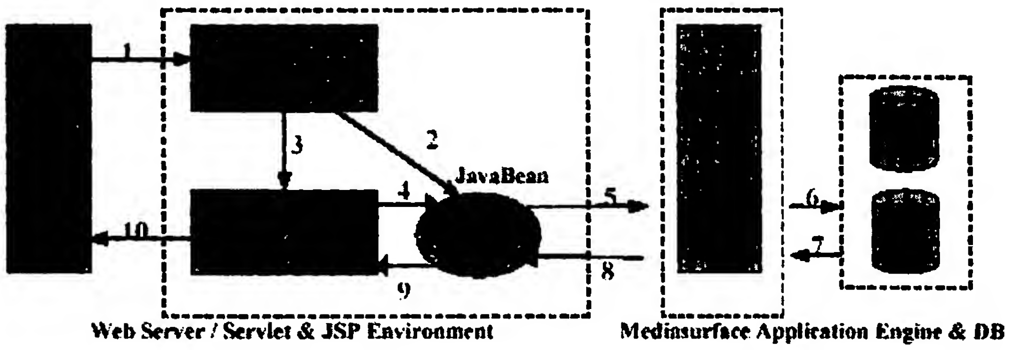
Content Management Console CMC
The Content Management Console is an administration layer allowing content contributors and editors to work on content over the web. The Content Management Console works with the Mediasurface Application Engine.
CMC is based on servlet and JSP's. **MediaSurface is ready to give us the sources** of this application so we will be able to personalize and develop specific features associated to ours needs.

MVC Introduction

A summary of MVC as used here can be written as: a 'controller' servlet handles the initial request from a browser, partially processes the data and sets up any required 'Model' Java Beans to encapsulate presentation logic and state. Finally the controller determines which 'view' JSP to forward the presentation of results to."

This approach typically results in the cleanest separation of presentation from content, leading to separation of roles and responsibilities of the developers and page designers.

Figure below highlights not only the structure of the MVC style of application, but also the process flow resulting from a standard browser request. The Web server in Figure below executes the presentation logic according to the MVC paradigm while communicating with the domain model as represented by the MAe executing on an Application Server.


1
2
3
4
5
6
7
8
9
10
JavaBean
Web Server / Servlet & JSP Environment
Medinsurface Application Engine & DB

MVC Process Flow Diagram

Description of Steps Within a Standard Process
The architecture diagram shown in Figure above contains numerals that show the process steps involved in a typical browser request to the application. The details of these steps are:

1. Browser Request – A user makes a request to the application by clicking on a URL or submitting a form. The servlet breaks down the request and processes any input.

2. Bean Instantiation – If the request demands access to any of the Java Beans, then the Java Beans are instantiated from within the servlet (unless they already exist).

3. Redirection to JSP – The process flow is passed to a JSP to handle the presentation.

4. Request of Bean Property – The JSP may now refer to the properti s of th Java Beans, which hav been instantiated earlier in the s rvlet.

5. Request of API Prop rty – The Java Bean uses the API to

retrieve th required data from the MAe based on th property request.

6. Request of Data from Database –The MA accesses th relevant tables in the database based on the property request.

7. Retrieval of Data from Database – The data is sent from the database to the MAe.

8. Retrieval of API Property – The API shapes the data appropriately and sends the reshaped data to the Java Bean. Using the Mediasurface Java API

9. Retrieval of Bean Property – The Bean shapes the data appropriately and sends the reshaped data to the JSP in a format that can be rendered by a web browser.

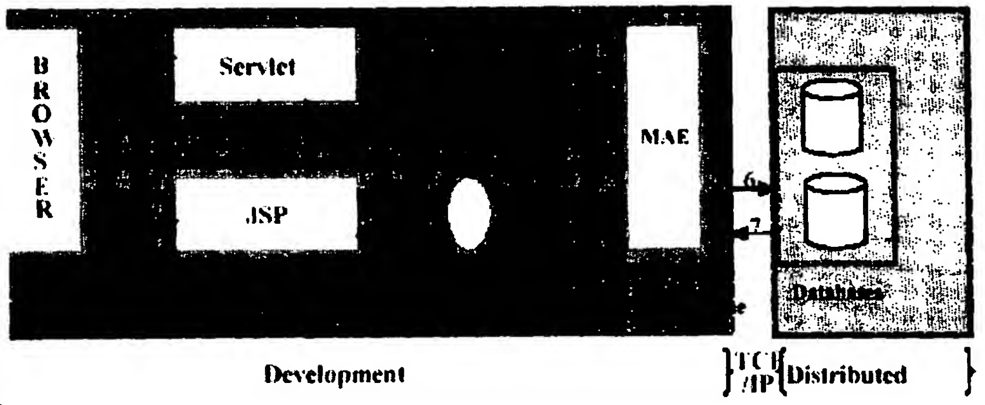
10. Dispatch of Response to Browser – After the HTML output is complete, the resulting response is sent to the browser.

Physical Distribution of Tiers

Figure below shows that the application is divided into four logical tiers:
1. Client (Browser)
2. Web Server
3. Application Server
4. Enterprise Servers / Data Sources

For development, these logical tiers can be reduced to two physical tiers:
1. Development Machine – contains the client, the web server and the Mediasurface Java
Application Engine
2. Database Machine – runs the Oracle RDBMS

Figure below shows the separation of logical and physical tiers.


BROWSER
Servlet
JSP
MAE
6
7
Database
Development
TCP/IP
Distributed